PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº 21 - Setembro de 2009

www.portugal-a-programar.org

Diagrama de Blocos

Templates T4

LUA: Linguagem de Programação

Acesso a BDs remotas em Windows Mobile 6

ISSN | 1647-0710

índice

equipa PROGRAMAR

coordenadores
Joel Ramos
Pedro Abreu

editor
António Silva

capa
Maurício Rodrigues

redacção
Augusto Manzano
Bruno Oliveira
Jorge Paulino
Miguel Alho

equipa de revisão
Bruno Oliveira
Fernando Martins
Miguel Rentes
Nuno João

equipa de divulgação
David Ferreira

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Regresso ao Trabalho

Voltamos a Setembro, mês do inevitável regresso às aulas para os estudantes portugueses, após as prolongadas e muito apreciadas férias de Verão. Subitamente, milhares de jovens vêem-se envolvidos numa corrida à preparação para as aulas, para muitos o famoso primeiro contacto com o Ensino Superior, as candidaturas, matrículas, e por vezes a adaptação a uma nova cidade. Todo este repentino movimento após as férias custa e não é propriamente agradável pensar no ano de trabalho que se avizinha, mas para projectos como o nosso, que vivem principalmente da vontade e disponibilidade de estudantes voluntários, este é o regresso à normalidade.

É natural que o Verão seja a época baixa no que diz respeito a qualquer tipo de actividade extra-profissional. No caso particular da revista, grande parte da equipa, incluindo redactores, revisores, colaboradores e mesmo coordenadores, esteve pouco disponível durante a preparação desta edição, o que nos obrigou a adiar o lançamento um mês, tal como no Verão passado - adiamento que lamentamos e quanto ao qual esperamos compreensão dos leitores. Contrariando a pouca participação, destacamos e agradecemos ao nosso editor António Silva, pela forma como conseguiu, de forma autónoma e praticamente individual, transformar um conjunto de artigos em mais uma edição que esperamos estar à altura das anteriores.

Se, por um lado, podemos agradecer os esforços dos que construíram esta edição e têm permitido manter este projecto mais ou menos regular, por outro as dificuldades que enfrentamos nestes períodos de férias exigem reflexão. Não há forma de garantir que uma edição terá participação suficiente, e são momentos como este que nos mostram quão abalável é um projecto desta dimensão que julgamos sólido, pelo simples facto de sermos uma equipa de voluntários e de não haver quaisquer obrigações por parte dos colaboradores. Provavelmente este é um problema comum a todos os projectos deste tipo que exigem actividade regular, mas é importante minimizá-lo e encontrar alternativas que nos permitam evitar o fracasso.

É simples apontar o que há a fazer. Preparar cada edição exige participação e trabalho de todo o tipo de colaboradores, e se grande parte se revela indisponível nestes períodos - indisponibilidade que não é, de forma nenhuma, condenável - é necessário alargar a equipa, procurando novos interessados em participar. A tendência a reduzir o ritmo de trabalho à medida que se prolonga a sua colaboração é outro factor que mostra a importância de promover a rotatividade e evitar a estagnação na equipa, introduzindo regularmente membros novos e motivados, produzindo mais e melhor conteúdo antecipadamente, para que se assegure a regularidade e a solidez do projecto face a estes obstáculos. E é por isso que a vossa vontade, a capacidade de darem o passo e passarem de leitores a colaboradores, é vital para este projecto. Colaborem.

Pedro Abreu

Dados de acesso à Internet guardados

Os dados dos acessos à internet por parte dos internautas ficam desde o dia 5 de Agosto guardados, ficando os ISP responsáveis por eles.

http://www.pplware.com/2009/08/05/dados-de-acesso-a-internet-ficam-guardados-a-partir-de-hoje/

Concurso de Projectos de Programação P@P

Iniciou-se a 5ª edição do concurso de projectos de programação, na categoria Software livre. Se tens um projecto a desenvolver ou já desenvolvido, e queres mostrar o que vales, participa no concurso! Não é necessário que seja uma aplicação completa e poderá ser apenas um controlo, componente, biblioteca, etc.

Ao participares neste concurso tens a oportunidade de mostrar os teus dotes de programação, competir com outras aplicações, ou simplesmente participar pelo prazer de programar!

Como prémio, temos uma tshirt do P@P para oferecer e um livro da editora FCA (http://www.fca.pt), isto para além do reconhecimento que a aplicação terá de toda a comunidade.

Vê tudo o que precisas de fazer em http://www.portugal-a-programar.org/concurso

Boa sorte para todos!

www.fca.pt

JavaPT09

Concebido como um fórum aberto e gratuito de carácter tecnológico, o JavaPT09 traz até si os principais especialistas da Sun e da indústria para o ajudar a explorar o potencial das tecnologias Java.

Saiba quais as últimas novidades, soluções e tendências das tecnologias Java. Participe no maior encontro da comunidade Java e tecnologias abertas em Portugal.

Onde: Universidade do Minho, Braga, Portugal

Quando: 17 de Setembro 2009

Quem deve Participar?

- Developers / Programadores
- Arquitectos de Software
- ISV's - Independent Software Vendors
- Universidades (Corpo Docente, Investigadores e Estudantes)
- Gestores de Projectos e Serviços
- Responsáveis de Desenvolvimento

http://pt.sun.com/sunnews/events/2009/sept/javapt09/index.jsp

Olimpíadas Internacionais de Informática 2009

1 dos 3 representantes de Portugal nas International Olympiad in Informatics (IOI), que decorreram na semana de 8 a 15 de Agosto na Bulgária, conseguiu a 2ª medalha para Portugal na história das IOI. Por isso, ficam desde já aqui expressos os parabéns públicos ao nosso coordenador Pedro Abreu de toda a equipa da revista PROGRAMAR, e também da comunidade Portugal-a-Programar pela excelente classificação e pela medalha recebida, fruto de um grande empenho e dedicação.

Fica também aqui expressos os parabéns para a restante comitiva pelo esforço em levar o nome de Portugal mais longe.

http://www.dcc.fc.up.pt/oni/

Movimentação de Controlos em Run-Time

Quando é necessário movimentar controlos com o rato, existem diferentes códigos que necessitam de cálculos, variáveis, etc, de modo a arrastar o controlo de um lugar para o outro, enquanto o botão do rato estiver pressionado.

Se é verdade que não é muito utilizado é verdade que quando é preciso não é um processo muito simples de fazer (pelo menos dá algum trabalho).

A forma mais simples de o fazer (que eu tenha conhecimento) é utilizando o método DefWndProc() que permite enviar mensagens para o sistema. É uma utilização de subclasses que irá fazer o mesmo que o Windows faz quando o utilizador pressiona o rato na barra de título e arrasta-o.

Ora isto permite simplificar uma série de coisas como arrastar um controlo (como por exemplo uma Label ou TextBox) ou mesmo o próprio Form. No caso do Form, e quando este não tem barra de titulo, dá imenso jeito.

Dois exemplos para uma TextBox e o Form:

```
' Constantes com a indicação de que o utilizador pressiona
' o botão esquerdo do rato e da barra de titulo do Form
Const WM_NCLBUTTONDOWN As Integer = &HA1
Const HTCAPTION As Integer = 2


' Movimentação da TextBox no Form
Private Sub TextBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseDown

    ' Caso esteja a ser pressionado o botão esquerdo
    If e.Button = Windows.Forms.MouseButtons.Left Then

        ' Liberta a captura do rato. Esta captura vai a True
        ' quando o rato é pressionado, e impede a movimentação
        TextBox1.Capture = False

        ' Envia uma mensagem que irá movimentar o controlo
        Dim msg As Message = Message.Create(TextBox1.Handle, WM_NCLBUTTONDOWN,_
                             New IntPtr(HTCAPTION), IntPtr.Zero)
        Me.DefWndProc(msg)

    End If

End Sub
```

É possível ainda utilizar outras constantes para fazer outras acções, como por exemplo utilizando a HTBOTTOMRIGHT (Const HTBOTTOMRIGHT As Integer = 17) em vez da HTCAPTION, que irá efectuar o resize do Form. A constante HTBOTTOMRIGHT é a indicação de que o utilizador está a fazer o redimencionamento do Form no canto inferior direito.

Inserir Imagens no SQL Server

Existem duas alternativas para a gestão de imagens: guardar apenas a localização e colocar a imagem no servidor ou gravar a imagem directamente num campo.

Para inserir a imagem é necessário ler, gravar num array de bytes e finalmente utilizar esse array para gravar na base de dados.

```
' Define a Connection String
Dim myConnectionString As String = _
    "Data Source=.\SQLEXPRESS;AttachDbFilename='c:\myDatabase.mdf';Integrated
Security=True;User Instance=True"

Try
    ' Cria um novo FileStream para leitura da imagem
    Dim fs As New IO.FileStream("c:\image.jpg", IO.FileMode.Open,
IO.FileAccess.Read)

    ' Cria um array de Bytes do tamanho do FileStream
    Dim ImageData(fs.Length() - 1) As Byte

    ' Lê os bytes do FileStream para o array criado
    fs.Read(ImageData, 0, ImageData.Length)

    ' Fecha o FileStream ficando a imagem guardada no array
    fs.Close()

    Using connection As New SqlClient.SqlConnection(myConnectionString)
        ' Define o commando Transact-SQL para inserir dados
        Dim SQL As String = "INSERT INTO contacts ([name],[img]) VALUES
(@name,@img);"
        Dim command As New SqlClient.SqlCommand(SQL, connection)

        ' Define os parametros para a inserçao de dados, onde está o array
        ' de bytes(imagem) a ser inserida. O tipo do campo é Image
        command.Parameters.Add("@name", SqlDbType.VarChar).Value = "jpaulino"
        command.Parameters.Add("@img", SqlDbType.Image).Value = ImageData

        connection.Open()

        ' Insere os campos no SQL
        command.ExecuteNonQuery()
    End Using

Catch ex As Exception
    MessageBox.Show(ex.Message, My.Application.Info.Title, MessageBoxButtons.OK,
MessageBoxIcon.Error)
End Try
```

Estás a ver o que não fizeste?
Esta página está em branco por tua culpa!
http://www.revista-programar.info/front/yourpage

Construção de Diagramas de Blocos

Introdução

Este artigo objectiva discutir o uso da norma ISO(International Organization for Standardization) 5807:1985 (E) na elaboração e documentação da representação gráfica do processo de raciocínio lógico utilizado no desenvolvimento da programação de computadores para linguagens de programação de computadores da primeira à terceira gerações, tomando-se por base os parâmetros apresentados na norma ISO 5807-1985 (E): Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts (Processamento de Informação – Documentação de símbolos e convenção para dados, diagramas de fluxo de programa e sistemas, diagramas de programas de redes e diagramas de recursos de sistemas).

Espera-se com esta proposta fornecer aos profissionais da área de desenvolvimento de programas de computadores subsídios para a elaboração padronizada e formal da representação gráfica da lógica de programação de computadores.

Histórico

A norma ISO 5807-1985 (E) para a definição e elaboração de diagramas de fluxos para a área de desenvolvimento e projecto de software é a consolidação de duas normas anteriores: ISO 1028(Information processing - Flowchart symbols) e ISO 2636 (Information processing - Conventions for incorporating flowchart symbols in flowcharts.), ambas publicadas em 1973. Em particular, a norma ISO 1028 foi editada a partir da norma ANSI (American National Standards) X3.5 de 1970.

Segundo informação da própria norma ISO 5807-1985 (E), a definição e elaboração de diagramas para a representação gráfica da linha de raciocino lógico a ser adoptada não deve restringir o uso de aplicações ou soluções particulares, uma vez que pode ocorrer a existência de vários tipos de soluções para os vários problemas de processamento de informação (ISO 5807, 1985, p. 1). Nota-se, com base no exposto, que a norma ISO 5807-1985 (E) sugere o uso de alguns critérios que devem ser adaptados segundo as necessidades existentes. Se de um lado, esta postura fornece a liberdade de trabalho desejada, por outro, acaba dando margem à existência de problemas de interpretação da própria norma.

Dentro deste aspecto e da liberdade de poder definir ou considerar situações particulares, é que este artigo propõe alguns padrões básicos na elaboração da representação gráfica, no sentido de que haja uma maior conformidade na sua elaboração e maior rigor na definição padronizada de sua apresentação visual. Dessa forma, ficará mais fácil a comunicação na elaboração de projectos, nos quais várias pessoas estarão envolvidas, mesmo que sejam de regiões geográficas diferentes.

É importante ressaltar que a norma ISO 5807-1985 (E) faz menção à definição de cinco tipos de representação gráfica: program flowcharts (diagrama de fluxo de programas, ou seja, diagramas de blocos, que é o tema deste artigo), data flowcharts (diagrama de fluxo de dados), system flowcharts (diagrama de fluxo de sistemas) program network charts (diagrama de programas de rede) e system resources charts (diagrama de recursos de sistema), dos quais apenas o primeiro tipo será discutido.

O conjunto de símbolos gráficos utilizados na elaboração de diagramas tem como principal objectivo demonstrar de forma clara a linha de raciocínio lógico utilizada por um programador de computadores, de forma que seja fácil a quem não efectuou a tarefa de programação entender o que se pretende que aquele programa faça.

Segundo (PRESSMAN, 1995, p. 453), "uma imagem vale mil palavras, mas é importante diferenciar qual imagem e quais mil palavras" se pretende realmente referenciar. A importância da representação gráfica da linha de raciocínio lógico é considerada também por BERG & FIGUEIRÓ (1998, p. 18), quando afirmam que as representações gráficas implicam acções distintas, deixando claro que "tal propriedade facilita o entendimento das ideias (...) e justifica a sua popularidade". A importância da representação gráfica da lógica de programação de um computador não é algo novo, pois já havia sido apresentada por GOLDSTEIN & VON NEUMANN (1947).

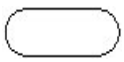
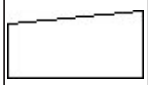
Verifica-se que, apesar da concordância de alguns autores no sentido de reconhecer o valor do instrumento gráfico na representação da linha de raciocínio lógico de um programa, não existe uma definição padronizada para o seu desenvolvimento, o que acaba por gerar diversas outras formas de definição e por conseguinte, leva a incorrer em muitos erros de definição. PRESSMAN (1995, p. 453) adverte que "se as ferramentas gráficas forem mal utilizadas, a figura errada pode levar ao software errado".

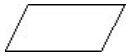
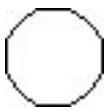
Parte da ocorrência da má interpretação e uso das

ferramentas de representação gráfica decorrem do facto de a norma ISO 5807-1985 (E) ser um instrumento apenas informativo e não regulador do processo de criação e uso dos símbolos gráficos nos projectos de desenvolvimento da lógica de programação a ser utilizada.

Simbologia gráfica

Na figura 1 são apresentados apenas os símbolos mais importantes da norma ISO 5807-1985 (E) , que representam a sequência de operações de um programa e de acesso ao caminho dos dados na definição da estrutura gráfica de um diagrama de blocos.

Símbolo	Significado	Descrição
	Terminal Terminator	Este símbolo representa a definição de início e fim do fluxo lógico de um programa (ISO 5807, 1985, p. 9). Também é utilizado na definição de subrotinas de procedimento ou função.
	Entrada manual Manual input	Este símbolo representa a entrada manual de dados, normalmente efectuada em um teclado conectado directamente ao console do computador (ISO 5807, 1985, p. 3).
	Processamento Process	Este símbolo representa a execução de uma operação ou grupo de operações que estabelecem o resultado de uma operação lógica ou matemática (ISO 5807, 1985, p. 3).
	Exibição Display	Este símbolo representa a execução da operação de saída visual de dados em um monitor de vídeo conectado ao console do computador (ISO 5807, 1985, p. 3).
	Documento Document	Este símbolo representa a execução da operação de saída de dados em um documento emitido por uma impressora na forma de relatório (ISO 5807, 1985, p. 3).

Símbolo	Significado	Descrição
	Dados Data	Este símbolo representa oficialmente dados de uma forma genérica (ISO 5807, 1985, p. 2). Tipicamente é associado a operações genéricas de entrada e saída de dados, desde que identificados.
	Decisão Decision	Este símbolo representa o uso de desvios condicionais para outros pontos do programa de acordo com situações variáveis (ISO 5807, 1985, p. 4).
	Preparação Preparation	Este símbolo representa a modificação de instruções ou grupo de instruções existentes em relação à acção de sua actividade subsequêncial (ISO 5807, 1985, p. 4).
	Processo predefinido Predefined process	Este símbolo representa definição de um grupo de operações estabelecidas como uma subrotina de processamento anexa ao diagrama de blocos (ISO 5807, 1985, p. 4).
	Conector Connector	Este símbolo representa a entrada ou saída em outra parte do diagrama de blocos. Pode ser usado na definição de quebras de linha e na continuação da execução de decisões (ISO 5807, 1985, p. 9).
	Linha tracejada Dashed line	Este símbolo representa uma alternativa na definição do relacionamento entre duas operações do diagrama de blocos. Também pode ser usado na definição de área de comentários (ISO 5807, 1985, p. 7).
	Linha Line	Este símbolo representa a acção de vínculo existente entre os vários símbolos de um diagrama de blocos. Normalmente possui a ponta de uma seta indicando a direcção do fluxo de acção (ISO 5807, 1985, p. 6).

É importante notar que para a definição do desenho de um diagrama de blocos pode-se utilizar um gabarito (figura 1) que normalmente possui o conjunto de símbolos básicos necessários para o devido uso. Os gabaritos possuem também, dependendo do modelo, alguns símbolos que não estão definidos na norma ISO 5807-1985 (E) ou, muitas vezes, não apresentam símbolos que estão definidos na norma ISO 5807-1985 (E).

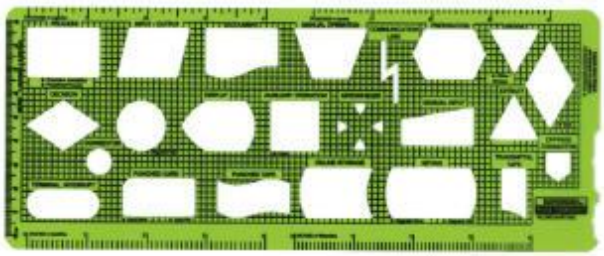
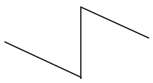
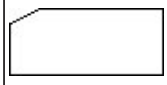


Figura 1 – Gabarito para o desenvolvimento de diagramas de blocos padrão IBM

Para melhor ilustrar esta abordagem, serão apresentados os demais símbolos existentes no gabarito representado pela figura 2, os quais não foram comentados anteriormente, mas que são previstos de uso, segundo a norma ISO 5807-1985 (E). A figura 3 mostra os referidos símbolos.

Símbolo	**Significado**	Descrição
	Operação manual Manual operation	Este símbolo representa a execução de qualquer operação que poça ser realizada por seres humanos, e que não estejam previstas em nenhum outro símbolo (ISO 5807, 1985, p. 4)
	Linha de comunicação Communication link	Este símbolo representa a transferência automática de informações ou dados entre locais diferentes, por meio de linhas de comunicação (ISO 5807, 1985, p. 7).
	Fita perfurada Punched tape	Este símbolo representa o uso de fita de papel ou fita plástica utilizada como um meio de armazenamento de programa e dados. Mecanismo de armazenagem de pouco uso (ISO 5807, 1985, p. 3).

Símbolo	Significado	Descrição
	Cartão Card	Este símbolo representa o uso de cartões perfurados de cartolina utilizado como meio de armazenamento de programa e dados. Mecanismo de armazenagem de pouco uso (ISO 5807, 1985, p. 3).
	Armazém de dados Stored data	Este símbolo representa a definição de acesso de abertura, fechamento, leitura e escrita em um determinado ficheiro de dados a ser realizada por um programa (ISO 5807, 1985, p. 2).

A figura 1 possui alguns símbolos que não foram apresentados nas tabelas, pois os mesmos não se encontram definidos na norma ISO 5807-1985 (E). De notar um importante detalhe em relação ao símbolo processo predefinido, que para ser desenhado utiliza como base o símbolo processo.

Sugestão de regras para complementação da norma ISO 5807-1985 (E)

A norma ISO 5807-1985 (E) apresenta alguns parcos exemplos de representações gráficas da forma de uso das cinco categorias de diagrama que aborda. Nenhum dos exemplos indicados aprofunda-se num uso mais formal dos símbolos. Com base nessa ocorrência, abre-se o espaço para a contribuição deste artigo.

Antes de exemplificar o uso específico de algumas situações relacionadas com o processo de representação gráfica da linha de raciocínio lógico de um programa de computador, é necessário considerar algumas convenções, baseadas na própria norma, bem como a adaptada a partir dela:

- os símbolos de identificação gráfica representam sempre uma operação ou conjunto de operações similares, podendo ser identificados por um rótulo relacionado com a própria acção do símbolo em uso, quando necessário;

- os símbolos devem ser conectados uns aos outros por linhas de setas que mostrem explicitamente a direcção do fluxo de programa a ser utilizado;

• a estrutura visual do diagrama de blocos deve a princípio estar orientada no sentido de cima para baixo, da direita para a esquerda. No entanto, dependendo da situação, este critério pode ser alterado, o que leva à necessidade de se manter as linhas de seta indicando a direcção do fluxo;

• a definição de inicialização e finalização de um diagrama de blocos ocorrerá com o uso do símbolo terminal, devidamente identificado;

• as operações de entrada de dados executadas manualmente serão genericamente representadas com o símbolo entrada manual;

• as operações de saída de dados serão genericamente definidas com o símbolo exibição, considerando o facto de estar em uso um monitor de vídeo. A excepção a esta regra ocorrerá aquando da definição da saída de dados em formato de relatório, definidas pelo símbolo documento;

• as operações de entrada e saída não definidas com os símbolos entrada manual, exibição e documento serão definidas com o símbolo "dados";

• a definição das variáveis nas operações de entrada e saída serão definidas nos símbolos apropriados. Quando houver mais de uma variável a ser utilizada, estas estarão separadas por vírgulas;

• as operações de processamento matemático e lógico estarão definidas com o uso do símbolo processo. Quando houver mais de uma operação a ser definida num mesmo símbolo, estas deverão estar separadas por linhas de acção;

• as operações de tomada de decisão para decisões simples, decisões compostas e laços condicionais serão representadas pelo símbolo decisão, que conterá internamente a definição da condição a ser avaliada logicamente;

• as operações de ciclo incondicionais (ciclos iterativos) serão representadas com o símbolo preparação, uma vez que este símbolo permite a realização de um grupo de tarefas relacionadas;

• em relação à definição de subrotinas será utilizado o símbolo processo pré-definido. Este símbolo deverá estar identificado com um rótulo que estará associado a outro diagrama de bloco inter-dependente ao programa e relacionado por um símbolo terminal;

• as operações que necessitarem do uso de conexão utilizarão o símbolo conexão. Este símbolo poderá ser usado na finalização de operações de decisões ou na identificação de vínculos entre partes de um programa e, nesse caso, deverão estar identificados com rótulos alfanuméricos;

• fica eleito o símbolo processo como um algo que pode mudar de valor , podendo representar qualquer acção lógica definida ou não, desde que a operação seja devidamente identificada por um rótulo descritivo. Excepções aos símbolos decisão e preparação que representam operações bem definidas não serão substituídos por qualquer outro símbolo.

Além das sugestões anteriormente propostas, poderão surgir ainda alguns pontos a serem definidos. Outros critérios que sejam necessários e que não foram aqui sugeridos deverão ser analisados e definidos em particular pela gestão de cada equipa de desenvolvimento.

Representações gráficas

Apresenta-se a seguir algumas situações ilustrativas em relação ao uso do instrumento gráfico, de forma a que possa vir a facilitar a montagem visual do processo da linha de raciocínio da lógica de programação a ser utilizada.

Não é foco deste artigo discutir a funcionalidade do princípio de execução lógica de um computador e de seus detalhes de operação. Objectiva-se apenas demonstrar como representar graficamente as acções lógicas de um computador na forma mais abrangente possível, de maneira que se possa utilizar a técnica de construção de diagramas com qualquer tipo de linguagem de programação de computadores.

A figura 2 apresenta um modelo de execução sequencial de passos a serem processados por um computador. Nota-se que para esta representação gráfica se utiliza o símbolo terminal com a identificação das acções de início e fim de um programa de computador. O símbolo processo que é usado na figura 4 ocorre de forma genérica representando qualquer tipo de operação.

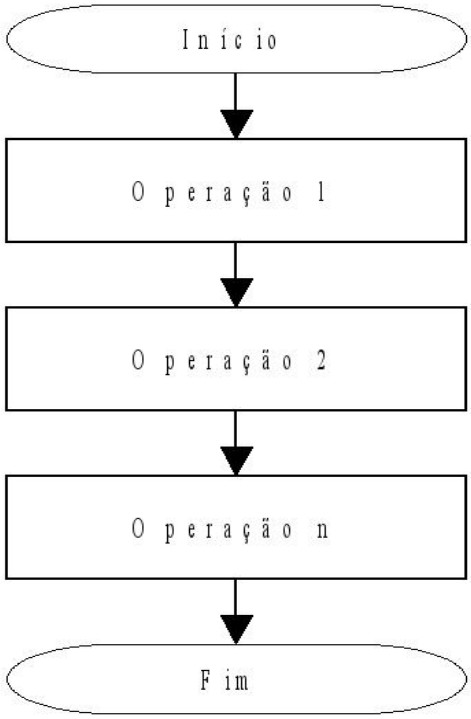


Figura 2 – Representação de uma sequência simples de operações.

A identificação da operação de início e fim de um diagrama de bloco é obrigatória, pois o símbolo terminal também é utilizado para representar o início e o fim da execução de uma sub-rotina. Assim sendo, torna-se necessário identificar correctamente a sua acção. A identificação utilizada no símbolo terminal ou demais símbolos poderá ocorrer em qualquer idioma.

Outro ponto a ser considerado é a definição das linhas de ligação representadas por setas, as quais possuem a finalidade de indicar a direcção que o fluxo do programa deverá seguir, além de estabelecer o vínculo de acção entre as várias operações que um programa de computador deve executar.

A figura 3 caracteriza a acção de uma tomada de decisão simples. Nesse caso, quando a condição for verdadeira, ocorrerá o desvio do fluxo do programa para a acção identificada como verdadeira e posterior acção de encontro com o símbolo conector. No caso de a condição ser falsa, o desvio do fluxo do programa ocorrerá pelo lado identificado como "não", o qual automaticamente irá ao encontro do símbolo conector para que o fluxo do programa tenha continuidade. Observe o uso do símbolo conector para estabelecer o ponto de continuidade do fluxo do programa.

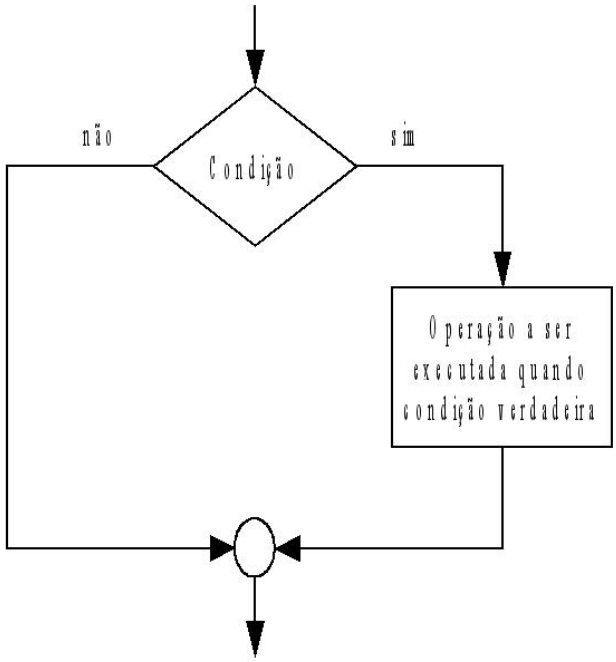


Figura 3 – Representação de uma tomada de decisão simples.

A figura 4 caracteriza a acção de uma tomada de decisão composta. Nesse caso, quando a condição for verdadeira ocorrerá o desvio do fluxo do programa para a acção identificada como verdadeira. Caso a condição seja falsa, será executada a acção identificada como falsa.
Torna-se fundamental em operações de tomada de decisão simples ou composta, identificar qual o lado do diagrama que representa a acção verdadeira e a acção falsa. Isto é necessário, uma vez que não existe um lado pré-definido para a acção verdadeira ou falsa.

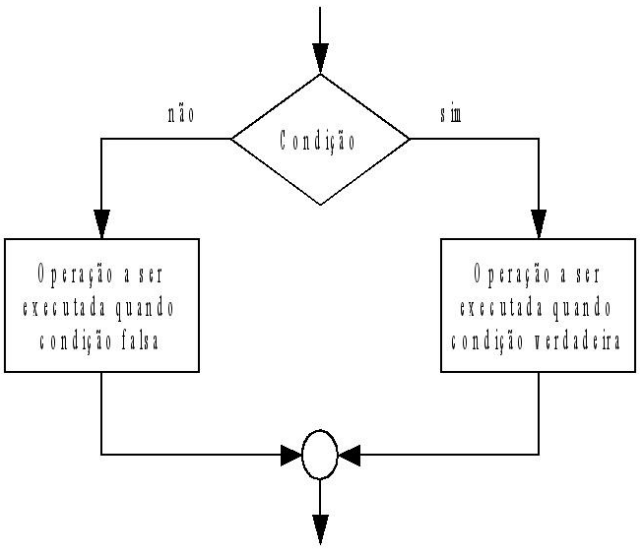


Figura 4 – Representação de uma tomada de decisão composta.

Torna-se fundamental em operações de tomada de decisão simples ou composta identificar qual o lado do diagrama que representa a acção verdadeira e a acção falsa. Isto é necessário, uma vez que não existe um lado predefinido para a acção verdadeira ou falsa.

A figura 5 apresenta o formato do ciclo de repetição condicional do tipo while...do. Este tipo de ciclo executa as operações a ele subordinadas enquanto a condição for verdadeira. Após a condição se tornar falsa, o ciclo é encerrado e o fluxo de acção do programa continua sua execução fora do ciclo de repetição.

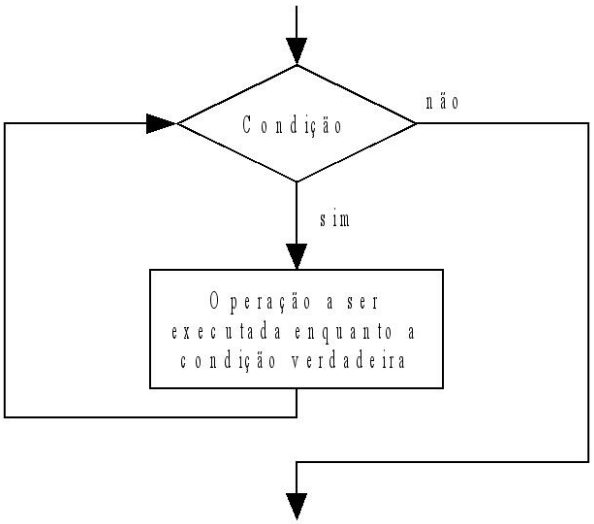


Figura 5 – Representação de ciclo condicional do tipo enquanto...faça.

A figura 6 apresenta o formato do ciclo de repetição condicional do tipo do...while. Esse tipo de ciclo executa as operações a ele subordinadas até que a condição seja verdadeira. Após a condição se tornar verdadeira, o ciclo é encerrado e o fluxo de acção do programa continua sua execução fora do ciclo de repetição.

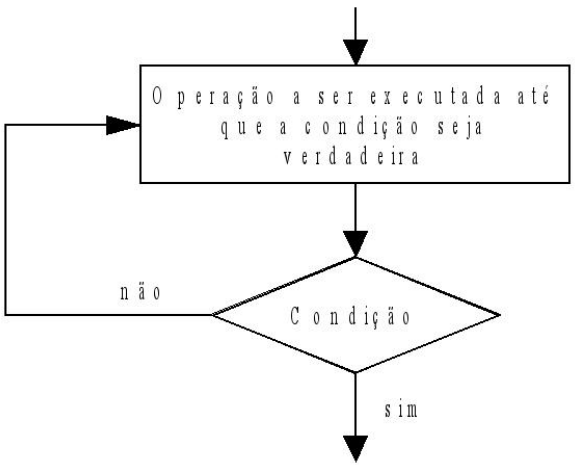


Figura 6 – Representação de ciclo condicional do tipo repita...até que.

A figura 7 apresenta o formato do ciclo de repetição incondicional do tipo for. Este tipo de ciclo executa as operações a ele subordinadas por uma variável de controlo iniciada por um valor, finalizado por outro valor diferente do valor inicial, podendo ser maior ou menor. A operação de incremento ou decremento é efectuada por um contador em cada passo de execução. Supondo um contador crescente de 1 a 10 com passo 1 positivo, sugere-se escrever na parte interna do símbolo preparação a descrição I ÿ 1, 10, 1; no qual I é a variável contador e as demais informações são respectivamente a inicialização da contagem, a finalização da contagem e o incremento entre o início e o fim da contagem. Se for utilizado um contador decrescente de 10 a 1 com passo 1 negativo sugere-se utilizar a descrição I ÿ 10, 1, -1.

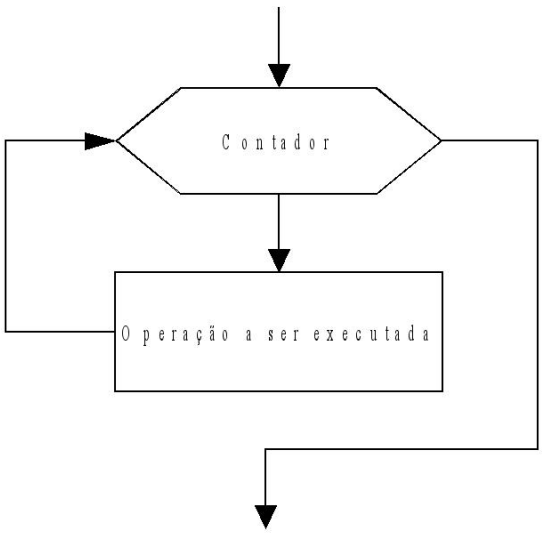
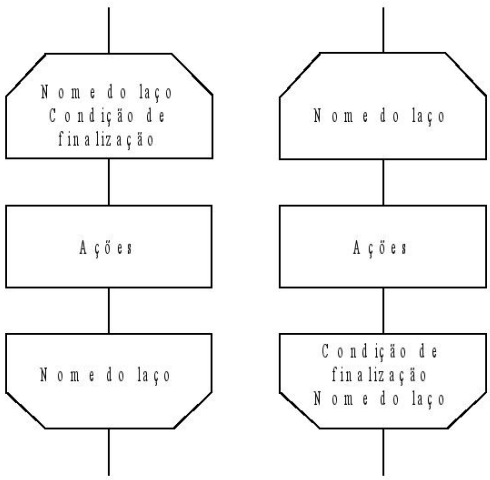


Figura 7 – Representação de ciclo incondicional do tipo para.

As figuras 5, 6 e 7 sugerem uma forma de representação de ciclos de repetição mais intimamente focada no contexto lógico da execução do código de programa num computador. No entanto, na norma ISO 5807-1985 (E), é sugerido o uso da estrutura gráfica representada pela figura 8.



A forma representada na figura 8 determina o uso da forma gráfica de representação dos ciclos do tipo while ou tipo for1 (desenho da esquerda) e do tipo do...while (desenho da direita). Nota-se apenas uma forma de representação gráfica sem o real vínculo com a representação do funcionamento mecânico dos tipos de ciclos. Representar os ciclos na forma mecânica, possibilita uma definição focada na funcionalidade do ciclo e não em uma mera representação do ciclo em si.

A figura 9 demonstra o critério de definição da chamada de uma sub-rotina com o uso do símbolo processo predefinido, na qual é indicado o nome da sub-rotina no programa principal. Ao lado direito encontra-se a definição da rotina secundária. Note as identificações utilizadas nos símbolos de terminal.

1 Para representar um ciclo do tipo for é necessário considerar no lugar da condição a definição de um contador.

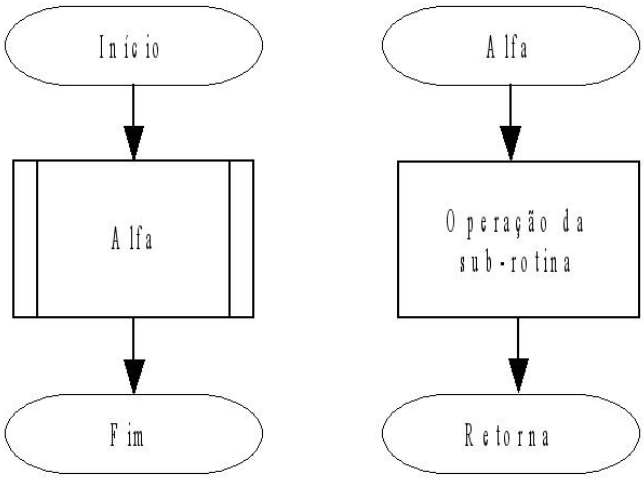


Figura 9 – Chamada e definição de uma subrotina simples.

A figura 10 demonstra o critério de definição da chamada de uma sub-rotina com o uso do conceito de passagem de parâmetro por valor, representado pelos dados indicados entre parênteses. Note no diagrama da direita a identificação no símbolo de terminal do nome da subrotina e dos parâmetros definidos entre parênteses.

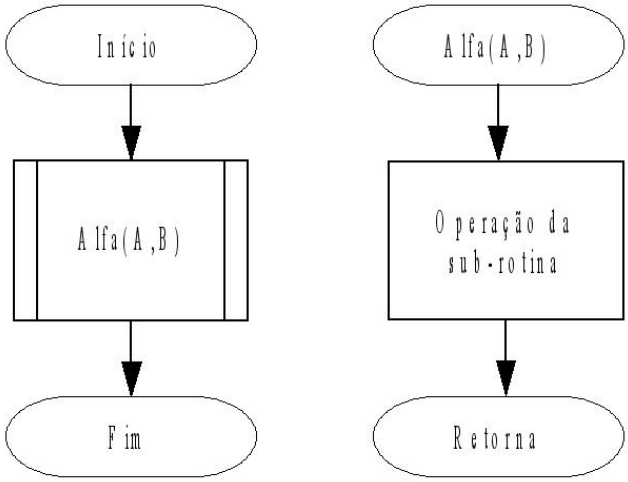


Figura 10 – Chamada e definição de uma subrotina com passagem de parâmetro por valor.

A figura 11 demonstra o critério de definição da chamada de uma sub-rotina com o uso do conceito de passagem de parâmetro por referência. Note no diagrama da direita, além da identificação no símbolo de terminal do nome da sub-rotina e dos parâmetros definidos entre parêntesis, a indicação da definição do rótulo Retorna(A,B), demonstrando a passagem de parâmetro por referência.

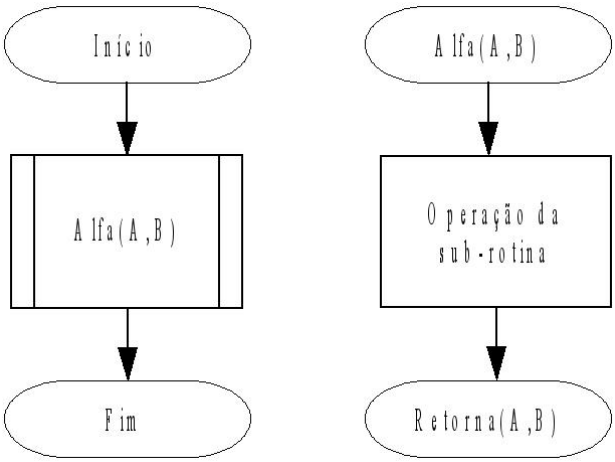


Figura 11 – Chamada e definição de uma subrotina com passagem de parâmetro por referência.

A figura 12 demonstra o critério de definição da chamada de uma sub-rotina de função, na qual é indicado o nome da função numa operação de processamento (por exemplo) e ao lado direito se encontra a definição da rotina secundária. Note as identificações utilizadas nos símbolos de terminal. Perceba que, ao indicar a finalização de uma função, se utiliza junto do rótulo de identificação Retorna o nome da própria função.

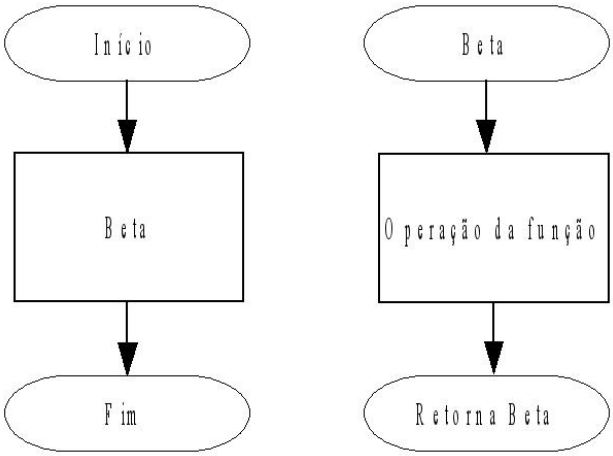


Figura 12 – Chamada e definição de uma função simples.

A figura 13 demonstra o critério de definição da chamada de uma função com o uso do conceito de passagem de parâmetro por valor, representado pelo dado indicado entre parêntesis. Note no diagrama da direita a identificação no símbolo de terminal do nome da função e do parâmetro definidos entre parêntesis.

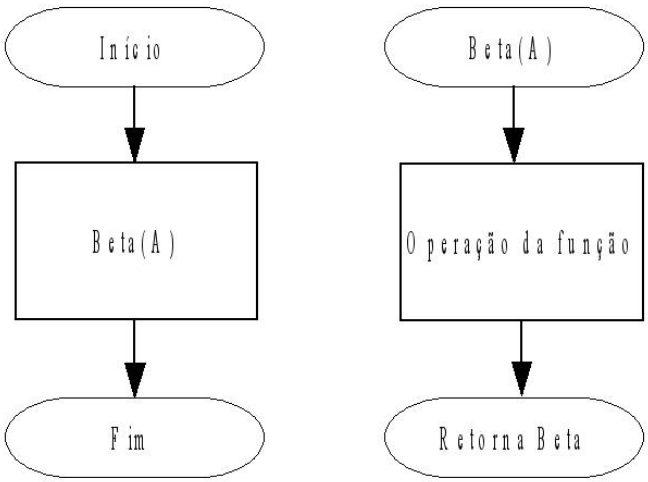


Figura 14 – Chamada e definição de uma função com passagem de parâmetro por referência.

A figura 15 representa visualmente o funcionamento lógico da estrutura de controlo lógico case existente em várias linguagens de programação de computadores. Esse tipo de estrutura lógica tem por finalidade desviar o fluxo de funcionamento do programa para uma acção e em seguida desviar o fluxo do programa para um ponto de conexão comum, representado pelo símbolo conector. Se a condição é válida (condição verdadeira), a acção definida após a condição é realizada e o fluxo de acção do programa é posicionado após a execução da acção estabelecida no ponto de conexão comum (representado pelo símbolo conector), que visa encerrar a acção da estrutura case. No caso de uma determinada condição ser falsa, o controlo do fluxo do programa é desviado para a condição seguinte, que repetirá o mesmo processo de funcionamento até chegar ao ponto de conexão comum.

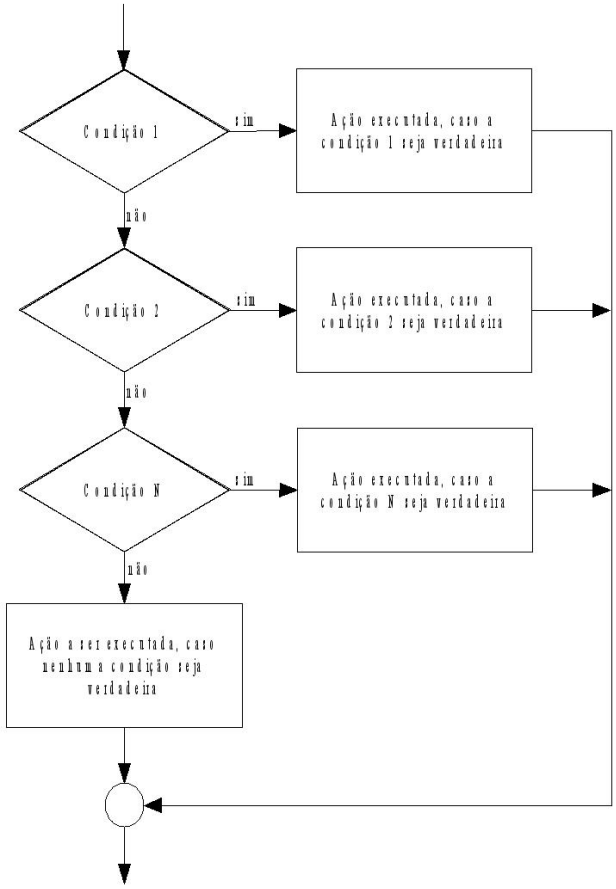


Figura 15 – Representação da estrutura de selecção case.

A figura 16 demonstra as operações de abertura e fecho de ficheiros de dados, utilizando-se o símbolo armazém de dados. Em casos que envolvam a manipulação de mais de um ficheiro de dados, utiliza-se sempre um símbolo de abertura e fecho para cada ficheiro em particular.

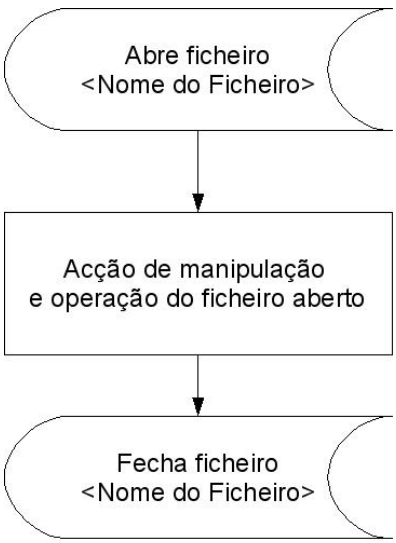


Figura 16 – Representação das operações de abertura e fecho de ficheiros.

A figura 17 apresenta um exemplo de conexão da linha de raciocínio lógico entre páginas, sugerido na norma ISO 5807-1985 (E), que dispensa qualquer comentário, pois apresenta-se de forma clara e objectiva. De notar que a indicação "A" dentro dos conectores é apenas uma referência ao ponto de conexão, podendo ser utilizada para esta representação outras letras e mesmo números. Outro detalhe com o uso de conectores é a possibilidade de representarem conexões na mesma página. Nesse caso, utiliza-se apenas os conectores, suprimindo a linha tracejada e a nota de descrição do ponto de conexão.

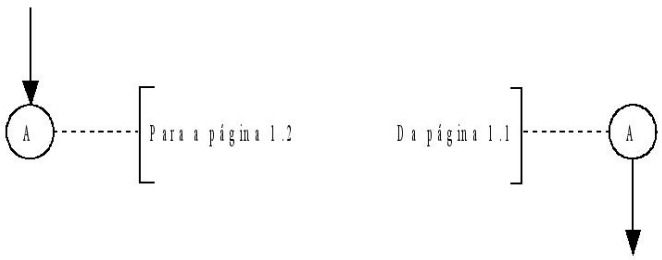


Figura 17 – Representação de indicações de conexão de páginas.

Elementos auxiliares de representação gráfica

É muito comum o uso de alguns elementos auxiliares de representação gráfica ao contexto lógico do desenvolvimento da programação de computador, principalmente no que diz respeito ao uso de operações lógicas e matemáticas. Nesse sentido, existem alguns operadores auxiliares, tais como:

- operadores aritméticos;
- operadores relacionais;
- operadores lógicos.

Além dos operadores aritméticos, relacionais e lógicos existentes, há também a definição do uso de valores lógicos, quando do uso de avaliação condicional ou processamento não matemático, tais como:

- verdadeiro;
- falso.

A norma ISO 5807-1985 (E) não faz menção em relação ao uso desses elementos auxiliares de representação gráfica. Assim sendo, cada profissional define critérios particulares, o que, muitas vezes, dificulta a comunicação entre elementos do mesmo grupo, ou em grupos distintos. Nesse sentido, é necessário definir um procedimento padrão para o uso desses elementos de forma mais organizada e padronizada, conforme se apresenta a seguir.

Operadores aritméticos

Os operadores aritméticos são responsáveis pela representação das operações matemáticas a serem realizadas numa acção de processamento de cálculo. A tabela seguinte sugere uma forma padrão para a representação de cada operador aritmético a ser utilizado.

Operador	Operação	Exemplo de operação		Tipo de Resultado
/	Divisão	5 / 2	2,5	real
div	Divisão	5 div 2	2	inteiro
⊗	Multiplicação	3 ⊗ 4	12	real ou inteiro
+	Adição	5 + 7	12	real ou inteiro
-	Subtracção	8 - 3	5	real ou inteiro
↑	Exponencial	2 ↑ 3	8	real ou inteiro
←	Atribuição	x ← 2	2	O valor atribuído

Tomando-se por base a fórmula de Bhaskara utilizada para efectuar o cálculo da equação de segundo grau, tem-se como sua definição a estrutura seguinte.

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$\Delta = b^2 - 4ac$$

A fórmula de Bhaskara (bem como qualquer outra fórmula matemática) deverá ser representada formalmente para efeito de documentação num diagrama de blocos, num formato de expressão aritmética a ser mencionada no símbolo de processamento, utilizando-se para a definição de cada parte da fórmula uma linha de texto.

```
DELTA ← B ↑ 2 – 4 ⊗ A ⊗ C
X1 ← (–B + DELTA ↑ (1 / 2)) / (2 ⊗ A)
X2 ← (–B – DELTA ↑ (1 / 2)) / (2 ⊗ A)
```

Note que, na forma de expressão aritmética, o conceito de x' (x linha) está representado pela definição da variável X1 e o conceito de x" (x duas linhas) está representado pela definição da variável X2.

A indicação da extracção da raiz quadrada é definida na forma exponencial, considerando-se como expoente o inverso do índice da raiz. E no sentido de representar a operação de exponencial está sendo aplicado o símbolo "ÿ".

O sinal de atribuição representa a acção de transferência do valor de uma operação a variável definida à sua esquerda. Este símbolo é usado em substituição ao símbolo "=", uma vez que o símbolo "=" é usado para definir uma relação lógica, como descrito no item sobre operadores relacionais. Quanto à representação gráfica no símbolo de processamento de um diagrama de bloco, as expressões poderão ser desenhadas separadamente (Figura 20a) ou no mesmo símbolo (Figura 20b), respeitando-se a definição de uma linha de texto para cada expressão aritmética.

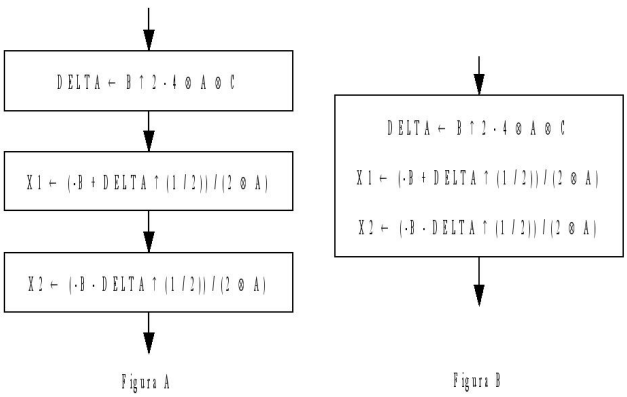


Figura 18 – Forma de representação de uma operação matemática.

Operadores relacionais

Os operadores relacionais são utilizados no estabelecimento das relações lógicas existentes entre dois elementos de uma determinada condição para tomadas de decisão efectivas ou da execução de ciclos de repetição lógicos. A tabela a seguir descreve os operadores existentes:

Simbolo	Significado
=	igual a
<>	diferente de
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que

O uso de um operador relacional será sempre definido no símbolo de decisão num diagrama de blocos, obedecendo à estrutura sugerida na figura 19.

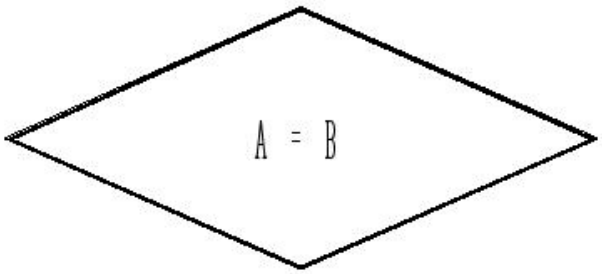


Figura 19 – Representação do uso de operadores relacionais em uma condição.

Os símbolos de representação dos operadores relacionais (=, <>, >, <, >=, <=) devem ser utilizados como descritos anteriormente, independentemente da forma que uma determinada linguagem de programação de computadores faça uso. É importante considerar que a definição de diagramas de blocos deve ocorrer de forma genérica e peculiar. Assim sendo, a codificação de um programa numa determinada linguagem de programação ocorre em função do modelo da diagramação definida, e não o contrário.

Operadores lógicos

Os operadores lógicos de conjunção (operador .e.) e de disjunção (operadores .ou. e .oue. – ou exclusivo) possuem a capacidade de associar para uma avaliação lógica mais de uma condição para uma mesma tomada de decisão. O operador de negação (operador .não.) tem por finalidade representar a negação de uma determinada condição. Alguns exemplos de uso são apresentados na figura 20.

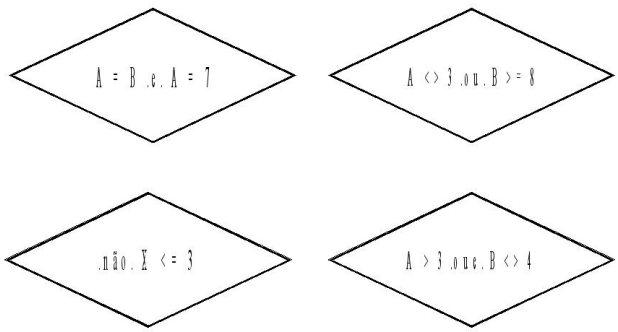


Figura 20 – Exemplos de uso de operadores lógicos.

Valores lógicos

Operações lógicas resultam em respostas verdadeiras ou falsas dependendo da condição lógica estabelecida. Nesse sentido, os valores lógicos falso e verdadeiro deverão ser expressos, segundo um critério particular:

- valor verdadeiro – representado pelos rótulos .VERDADEIRO. ou .V.
- valor falso – representado pelos rótulos .FALSO. ou .F.

O tipo de valor lógico explícito (.VERDADEIRO., .V., .FALSO. ou .F.) é normalmente utilizado em operações de processamento para representar a atribuição de um determinado valor lógico a uma determinada variável. A figura 21 representa uma acção de definição de valor lógico a uma variável.

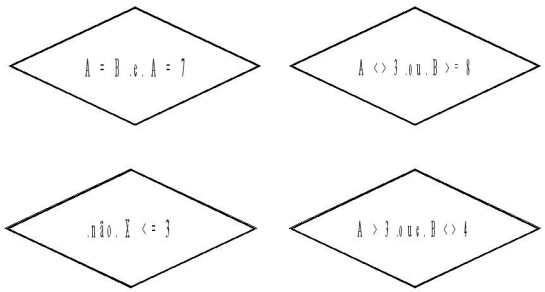


Figura 21 – Atribuição de um valor lógico.

O uso de valores lógicos também poderá ocorrer em situações de verificação de uma condição lógica na execução de uma tomada de decisão ou acção de execução de um ciclo de repetição. A figura 22 mostra um exemplo de uso de valor lógico numa condição.

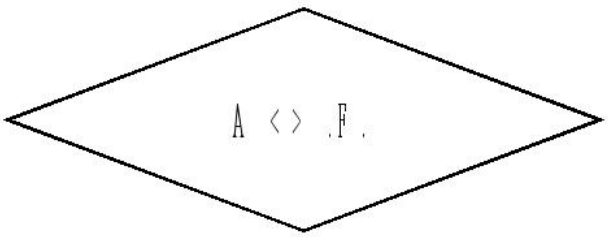


Figura 22 – Definição de valor lógico em uma condição

Nomenclatura

Em relação ao uso de diagramas no processo de representação gráfica da linha de raciocínio utilizada por um programador, torna-se ideal formalizar uma nomenclatura (já utilizada em parte por alguns poucos profissionais) de identificação dessas estruturas.
A figura 2 representa esquematicamente a estrutura de diagrama de bloco, o qual representa o processo sequencial de acções contínuas existentes entre as definições de início e fim da execução de um programa de computador.
As figuras 3, 4, 5, 6 e 7 representam esquematicamente a estrutura de diagrama de blocos, os quais representam a acção da implicação de um desvio condicional (figuras 2, 3, 4 e 5) ou um desvio não condicional (figura 7). Tanto um desvio condicional como um desvio incondicional, efectuam um desvio momentâneo do fluxo central do programa, para que um outro bloco adjacente de acções seja executado. Ao final da execução das acções desse bloco adjacente o programa retorna automaticamente para o fluxo central de funcionamento.
As figuras 9, 10, 11, 12, 13 e 14 representam esquematicamente a estrutura de diagramas de blocos (uso mais comum e provável) ou diagramas de bloco (uso mais raro, quase impossível). Será considerado uma estrutura de diagramas de blocos, quando existir um conjunto de diagramas definidos e relacionados para um mesmo problema computacional, onde esse conjunto envolva a definição de vários diagrama de blocos. O conceito diagramas de bloco é mais improvável de ser utilizado, pois dificilmente existirá um conjunto de diagramas com definições de operações meramente sequenciais no estilo apresentado na figura 2. As formas mais comuns e utilizadas são: diagrama de bloco e os diagramas de blocos.

Bibliografia

BERG, A. C. & FIGUEIRÓ, J. P., Lógica de Programação, Canoas, 1998, ULBRA, 168p.
PRESSMAN, R. S., Engenharia de Software, São Paulo, 1995, Makron Books, 1056p.
GOLDSTEIN, H. H. & VON NEUMANN, J. Planning and Coding Problems of an Electronic Computing Instrument. New York, In A. H. Taub (Eds.), von Neumann, J., 1947, Collected Works, McMillan. pp. 80-151.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem 23 anos de experiência em ensino e desenvolvimento de programação de software. É professor da rede federal de ensino no Brasil, no Centro Federal de Educação Tecnológica de São Paulo. É também autor, possuindo na sua carreira mais de cinquenta obras publicadas.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

Acesso de BDs remotas em Windows Mobile 6

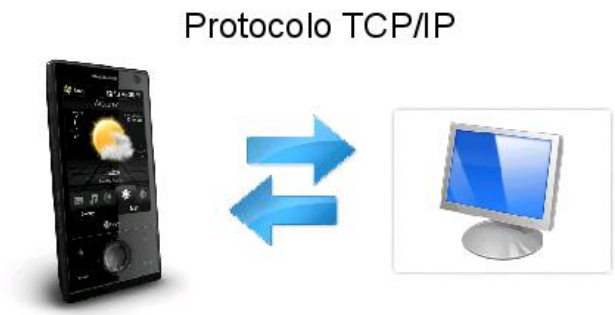
Concebido para funcionar em dispositivos móveis, o Windows Mobile é um dos sistemas operativos mais utilizados pelos fabricantes nos seus produtos.
Como é hábito, a Microsoft disponibiliza um conjunto de ferramentas muito interessantes para desenvolver aplicações para os seus produtos, e o Windows Mobile não é excepção. Neste artigo vamos abordar ligações a bases de dados remotas com o Windows Mobile, ou seja, vai ser descrito em detalhe como efectuar uma ligação de um dispositivo equipado com Windows Mobile com uma base de dados num dispositivo remoto. Isto é especialmente útil, se por exemplo pretendermos sincronizar informação entre um dispositivo móvel e um computador.

Pré-requisitos

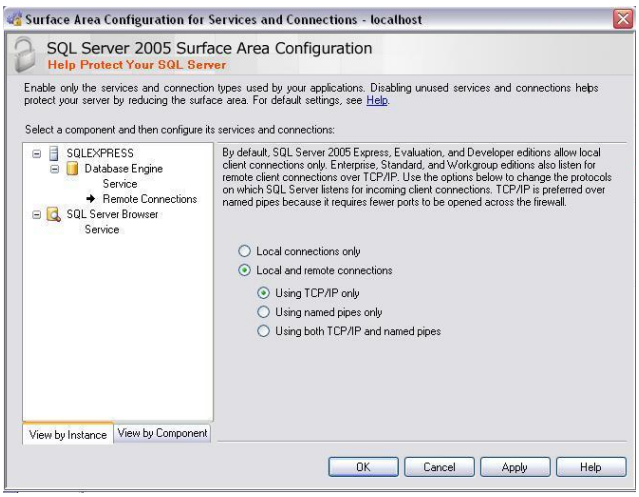
Antes de colocarmos mãos à obra, necessitamos de ferramentas para trabalhar. Começamos então, por efectuar o download do Microsoft SQL Server 2005 Express Edition no site da Microsoft (também pode ser a versão profissional) (http://www.microsoft.com/downloads/details.aspx?FamilyID=220549b5-0b07-4448-8848-dcc397514b41&displaylang=pt-br). Na instalação da aplicação, quando surgir a seguinte janela, seleccione Modo Misto (autenticação do Windows e autenticação do sql server), e introduza uma password à sua escolha. A partir deste ponto, apenas necessita de seguir as instruções do instalador.

Após a conclusão da instalação, necessita de instalar uma aplicação que permita, de uma forma visual, interagir com as bases de dados que vai criar mais à frente. Para isso, instale o Microsoft SQL Server Management Studio Express (http://www.microsoft.com/Downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en#filelist) e proceda á respectiva instalação.

Após iniciar o SQL Server, em Authentication deverá seleccionar SQL Server Authentication, e no login deve introduzir sa e a password que introduziu na instalação do SQL server, para então iniciar o Microsoft SQL Server Management Studio Express.
Para este teste, vamos criar uma base de dados com o nome teste, e uma tabela utilizador, com os campos id, nome com os tipo de dados int e varchar(50), respectivamente, e o campo id como chave primária. Introduza também alguns dados para teste.
Para que a ligação entre o dispositivo móvel e o computador se torne possível, vamos utilizar o protocolo TCP/IP. Para isso, necessitamos de configurar o SQL SERVER 2005 para permitir ligações remotas por TCP/IP.

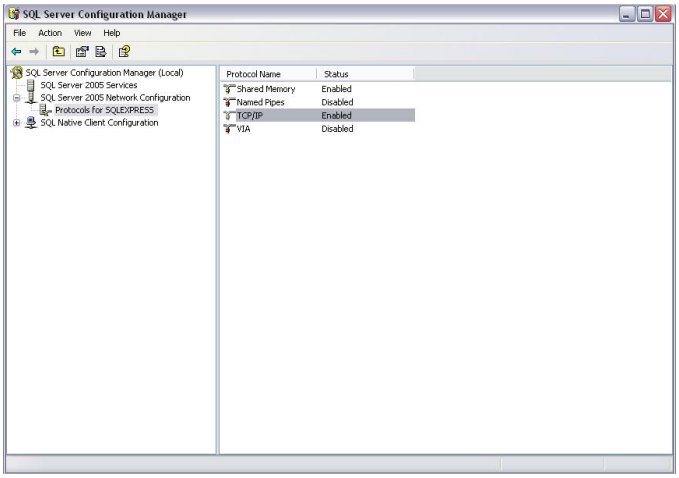


Vá a INICIAR -> PROGRAMAS -> MICROSOFT SQL SERVER 2005 -> CONFIGURATION TOOLS -> SQL SERVER SURFACE AREA CONFIGURATION. De seguida, seleccione: Surface Área Configuration for Services and Connections.

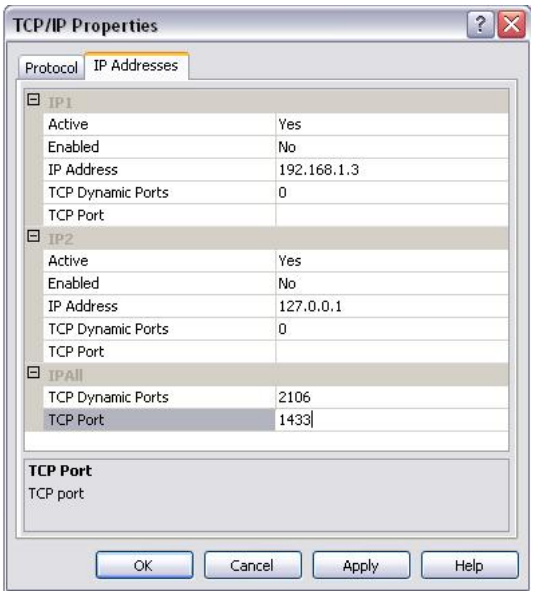


Nesta nova janela seleccione (à esquerda) Remote Connections e à direita a opção Local and remote connections -> using TCP/IP only. De seguida, clique em SQL Server Browser e certifique-se que a opção Startup type está em Automatic. Pressione Apply e seleccione Start, no botão correspondente. De seguida, seleccione o item Service do Database Engine e seleccione o botão stop seguidamente de start para reiniciar os serviços.

Neste exemplo, vamos utilizar a porta 1433 na ligação remota. Para configurar a porta dirija-se a INICIAR -> PROGRAMAS -> MICROSOFT SQL SERVER 2005 -> CONFIGURATION TOOLS -> SQL SERVER CONFIGURATION MANAGER.



Seleccione Protocols (à esquerda) dentro de SQL Server 2005 Network Configuration e certifique-se que TCP/IP está ENABLE. Dê duplo clique em TCP/IP. Nesta nova janela, certifique-se que TCP/IP está em Enabled no separador Protocol. De seguida, seleccione o separador IP ADDRESSES e na última linha (TCP Port) e coloque a porta 1433 conforme demonstra a figura seguinte.



Por fim, reinicie o computador para que as alterações surtam efeito (ou reinicie novamente os serviços). Já possuímos o SQL SERVER configurado e a base de dados com informação suficiente para que possamos testar a ligação. Chegou então o momento de configurar o Visual Studio e começar a programar.
Nota importante: Se estiver a utilizar Windows Vista poderão surgir problemas de permissões, como por exemplo, não lhe ser permitido parar os serviços do SQL Server. Se tal acontecer, pode tentar reiniciar o computador ou então entrar como administrador, efectuar as configurações e voltar a entrar como um utilizador normal.

Configuração Visual Studio

- Visual Studio 2005/2008 Professional
- Microsoft® Visual Studio® 2005 Team Suite Service Pack 1

http://www.microsoft.com/downloads/thankyou.aspx?familyId=bb4a75ab-e2d4-4c96-b39d-37baf6b5b1dc&displayLang=en

- ActiveSync 4.5

http://www.microsoft.com/downloads/details.aspx?FamilyID=9E641C34-6F7F-404D-A04B-DC09F8141141&displaylang=pt-br

- .NET Compact Framework 3.5

http://www.microsoft.com/downloads/details.aspx?FamilyID=E3821449-3C6B-42F1-9FD9-0041345B3385&displaylang=en

- Mobile 6 Professional SDK (Professional SDK)

http://www.microsoft.com/downloads/details.aspx?FamilyID=06111A3A-A651-4745-88EF-3D48091A390B&displaylang=en

- Virtual PC

http://www.microsoft.com/downloads/details.aspx?FamilyID=04d26402-3199-48a3-afa2-2dc0b40a73b6&DisplayLang=en#filelist

Quando o sistema operativo utilizado for o Windows Vista, então a lista de software necessária varia ligeiramente:

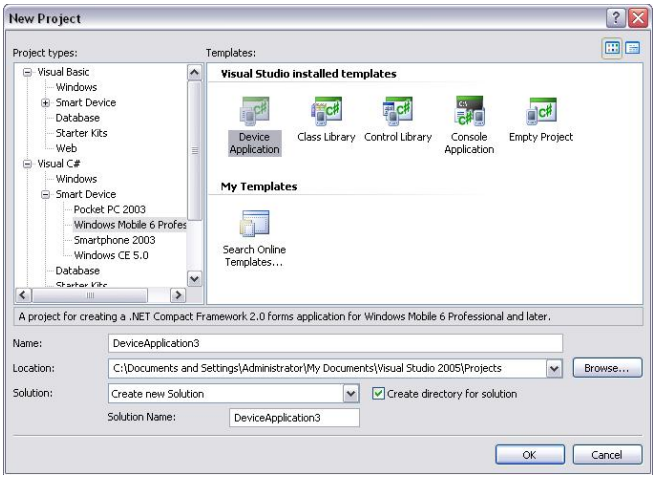
- Visual Studio 2005/2008 Professional com Service Pack para Windows Vista
- Mobile Device Center
- NET Compact Framework 2.0 com Service Pack 1
- Mobile 6 Professional SDK
- Virtual PC

De notar que a versão do Visual Studio 2005 tem de ser a Profissional e não a Express Edition fornecida gratuitamente pela Microsoft.

Elaboração do código

Após a instalação de todos os componentes referidos, estamos preparados para começar a programar em Windows Mobile 6.

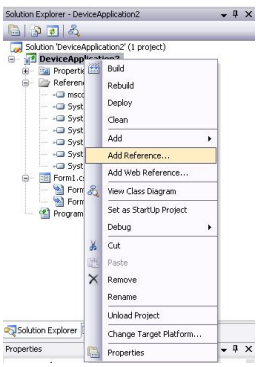
No Visual Studio, seleccionamos FILE-> NEW-> Project.



Nesta nova janela, seleccione Visual C#->Smart Device-> Windows Mobile 6 Professional e Device Application. De seguida, preencha os detalhes do projecto e clique em OK.

Nota: Neste exemplo vou utilizar a linguagem C# mas pode também utilizar Visual Basic.

Podemos agora visualizar o emulador e o formulário que vamos construir. Seleccione no separador vertical Toolbox e irá visualizar os componentes que pode arrastar para o formulário.
Vamos então construir a nossa aplicação, arrastando um botão ao qual vamos chamar Testar ligação. As propriedades dos botões podem ser alteradas recorrendo à caixa situada no canto inferior direito.
Antes de escrever o código necessitamos de adicionar as referências necessárias para a elaboração do exemplo. Na janela Solution Explorer, com o botão direito do rato, clique sobre o seu projecto e seleccione Add Reference.



Nesta nova janela seleccione System.Data.SqlClient no separador .NET e clique em OK.
Adicionada a referencia acrescente na sua classe using System.Data.SqlClient;.
Ligação à base de dados
De seguida, declare a variável Connection, e através do seguinte método teste a ligação:

```
private void TestaLigacao(String ip,
String porta,String nomeBd, String
username, String password)
{
        Connection = "Data Source=" +
ip + "," + porta + ";Initial Catalog="
+ nomeBd + ";User ID=" + username +
";Password=" + password + ";";
        SqlConnection comm = new
SqlConnection(Connection);
        try
        {
              comm.Open();
              comm.Close();

MessageBox.Show("Sucesso");
         }
         catch
         {
              MessageBox.Show("Erro na
ligação");
         }
}
```

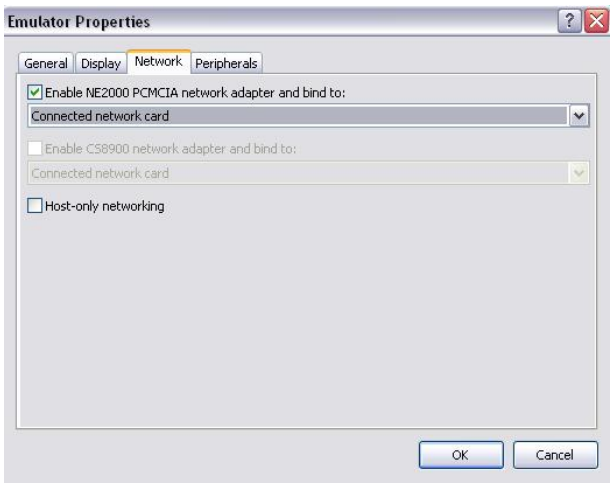
Este método recebe os parâmetros que necessita para efectuar a ligação, ou seja, o ip ao qual vamos conectar, a porta, o nome da base de dados, username e password. Estes dados são depois concatenados na string Connection, que vai ser a nossa string de ligação. Posteriormente criamos uma ligação SQL (SqlConnection comm = new SqlConnection(sConnection);) e testamos se a ligação é efectuada com sucesso.

No evento do botão que foi criado, invocamos o método descrito anteriormente:

```
private void button1_Click(object
sender, EventArgs e)
{

TestaLigacao("192.168.1.2","1433","test
e","sa","1234");
}
```

O primeiro parâmetro é o seu ip da rede, o segundo a porta de ligação que definiu, "teste" o nome da base de dados criada, "sa" o utilizador e 1234 é a password que coloquei quando instalei o SQL Server 2005 (não muito original diga-se).
Nota: Se não sabe qual o seu IP, vá ao menu iniciar -> executar e digite cmd. Quando surgir a janela do DOS digite o comando IPCONFIG e será exibida a informação sobre o seu IP.
Grave o projecto e execute através do botão start. Quando surgir a janela de escolha do emulador, pode escolher aquele que desejar, neste exemplo optei por Windows Mobile 6 Classic emulator.
Quando o emulador iniciar, seja paciente e aguarde que o emulador compile o programa. Quando o emulador é iniciado pela primeira vez, demora algum tempo a ficar preparado. Após surgir o seu formulário no emulador, certifique-se que o emulador tem acesso à internet, ou se tem IP da sua rede.
Para isso, na janela do emulador selecione File-> configure



No terceiro separador, certifique-se que a opção Enable NE2000 PCMIA network adapter and bind to está seleccionada e na caixa a opção Connected network card também o está, conforme demonstra a figura 15. Caso não esteja, coloque como descrito e reinicie o emulador.
Nota: a opção Connected network card, detecta automaticamente qual o hardware que está a utilizar para aceder à rede, mas também o pode escolher directamente.
Após reiniciar o emulador, no sistema operativo deste dirija-se ao seu programa (Start->programs-> File explorer-> program files-> nome do seu projecto). Nas opções de rede do emulador pode também especificar um IP para o emulador, se assim o desejar. Se tudo correu bem, irá visualizar a mensagem de sucesso após pressionar o botão Testar Ligação.
Agora que conseguimos ligar à base de dados vamos testar um Select,Insert, delete e update.

Altere o seu método TestaLigação (ou crie outro) de modo a que o método apenas abra a ligação e a atribua à variável comm global da classe. Neste exemplo, optei por criar um método novo chamado Abre_ligacao.

```
private void Abre_ligacao(String ip,
String porta,String nomeBd, String
username, String password)
{
        Connection = "Data Source=" +
ip + "," + porta + ";Initial Catalog="
+ nomeBd + ";User ID=" + username +
";Password=" + password + ";";
       comm = new
SqlConnection(Connection);
        try
         {
             comm.Open();
         }
         catch
         {
            MessageBox.Show("Erro na
ligação");
         }

}
```

SELECT:

```
private void TestaSelect()
{
    String query = "SELECT nome FROM
utilizador";
    SqlDataReader dr = null;
    try
    {
        SqlCommand cmd =
comm.CreateCommand();
        cmd.CommandText = query;
        cmd.ExecuteNonQuery();
        dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            String nome =
dr["nome"].ToString();
            MessageBox.Show(nome);
         }
      }
      catch
      {
          MessageBox.Show("Erro no
select");
      }
}
```

Este método selecciona o nome dos registos presentes na tabela utilizador e mostra-os um a um numa MessageBox, isto apenas a título de exemplo, pois estes dados podem ser colocados numa Listbox, ComboBox ou qualquer outro componente que permita listar dados. Crie agora um novo botão e invoque os métodos:

```
private void button2_Click(object
sender, EventArgs e){
     Abre_ligacao("192.168.1.2",
"1433", "teste", "sa", "1234");
     TestaSelect();
     comm.Close();
}
```

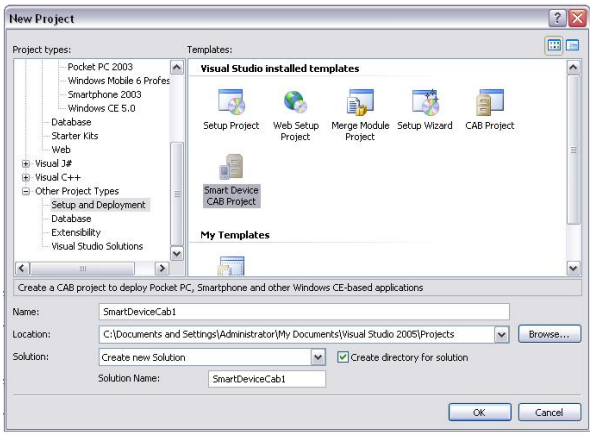
UPDATE:

```
private void TestaUpdate(){
   String query = "UPDATE utilizador
SET nome='Bruno Oliveira' WHERE
nome='Bruno'";
   SqlDataReader dr = null;
   try{
      SqlCommand cmd =
comm.CreateCommand();
      cmd.CommandText = query;
      cmd.ExecuteNonQuery();
MessageBox.Show("Dados actualizados
com sucesso");
   }
   catch{
       MessageBox.Show("Erro no
update");
   }
}
```

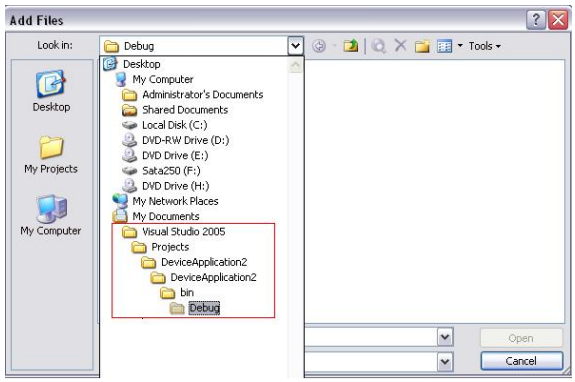
Para as queries DELETE e INSERT o procedimento é idêntico, alterando apenas a query, pois os comandos são mesmos.

Construir o executável

Agora que já conseguimos efectuar comandos sobre a base de dados, podemos testar o programa criado num dispositivo real. Para isso, no Visual Studio seleccione FILE -> NEW -> PROJECT -> Other project types -> Setup and Deployment e de seguida Smart Device CAB Project.



Clique com o botão direito do rato sobre Program files folder-> Add-> File. Dirija-se à pasta do seu projecto, vá até bin-> Debug e seleccione o ficheiro executável.



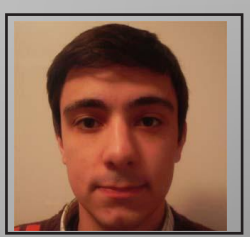
Por fim, seleccione Build Solution (CTRL+SHIFT+B). Dirija-se à pasta deste projecto que criou (CAB Project), e posicione-se dentro da pasta Debug.

Copiamos o ficheiro com o mesmo nome do projecto (o de maior tamanho) para a raiz do dispositivo utilizando o ActiveSync. Uma vez copiado o ficheiro, resta apenas instala-lo no dispositivo e testar a ligação.

Conclusão

Ao longo do artigo foi descrito, com bastante precisão, todos os passos necessários para a ligação a bases de dados remotas através de um dispositivo móvel. Caso tenha alguma dúvida, problema, sugestão ou crítica a este artigo, esteja à vontade de me contactar através do e-mail disponibilizado no final deste artigo.

SOBRE O AUTOR



Bruno Oliveira é estudante do 3o ano de EngenhariaInformática da ESTGF, bem como Trabalhador Independente na área de programação Web. Tem especial interesse pela área de Base de Dados e programação orientada a objectos.

bruno.oliveira@portugal-a-programar.org

Bruno Oliveira

Lua – Linguagem de Programação - Parte I

Este artigo faz parte de uma série de outros artigos que serão publicados sobre a utilização da linguagem de programação para computadores Lua. Neste artigo serão abordados assuntos relacionados com o surgimento da linguagem, quem a desenvolveu, utilização mundial e aquisição de alguns exemplos de programação sequencial.

Linguagem de Programação para Computadores Lua

A linguagem de programação para computadores Lua surgiu no ano de 1993 pelas mãos de Roberto Ierusalimschy (http://www.inf.puc-rio.br/~ roberto), professor associado do Departamento de Informática da PUC-Rio [http://www.inf.puc-rio.br] (Pontifícia Universidade Católica do Rio de Janeiro, Brasil) e consultor do Tecgraf/PUC-RIO (http://www.tecgraf.puc-rio.br) .
A linguagem foi desenvolvida a partir de uma necessidade interna do Departamento de Informática da PUC-Rio em desenvolver e atender uma parceria com a empresa PETROBRAS (http://www.petrobras.com.br) no sentido de fornecer suporte informático para as operações de escavação da empresa.
Após o atendimento deste projecto surgiram outros envolvendo a aplicação da nova linguagem de programação, como o desenvolvimento de um sistema de intranet para o Departamento de Informática da PUC-Rio em 1994. A partir de então a linguagem de programação Lua passou a ser utilizada em outros projectos como, por exemplo, o desenvolvimento dos jogos de computador pela empresa LucasArts (http://www.lucasarts.com/): Grim Fandango e Escape from Monkey Island.
A LucasArts foi a primeira empresa a fazer uso da linguagem no desenvolvimento de produtos. Posteriormente surgiu a Microsoft no desenvolvimento de jogos para a plataforma Xbox (UNIVERSIA, 2002).
A linguagem de programação Lua foi criada com o intuito de estender ,através de scripts, as aplicações de linguagens de programação mais pesadas, como o C ou o C++. A Lua tem sido utilizada no desenvolvimento de sistemas de Intranet, construção de sítios para Web e jogos electrónicos. No caso do desenvolvimento de jogos a linguagem é utilizada para auxiliar a execução de movimentos e diálogos entre as personagens. Desta forma, com a linguagem Lua é possível determinar que movimento será executado, enquanto que quem executa de facto o movimento é a linguagem à qual a linguagem Lua estará acoplada.
Como linguagem de programação caracteriza-se por ser rápida, poderosa, robusta, leve e distribuída no regime de software livre. Lua é uma linguagem de programação que combina uma simples estrutura sintáctica para o desenvolvimento de subrotinas com poderosas construções para descrição de dados baseados em tabelas. É uma linguagem que opera dados de forma dinâmica não necessitando da declaração do tipo de dados a ser utilizado, que são interpretados a partir de bytecodes para uma máquina virtual baseada em registradores.

Aquisição de um interpretador da Linguagem Lua 5.1

Para fazer uso da linguagem Lua, é de fundamental importância possuir instalado no computador o interpretador da linguagem. Para adquirir o programa, basta aceder ao endereço do sítio http://luabinaries.luaforge.net/. Lá, seleccione o menu Download e na lista apresentada seleccione a opção desejada vinculada ao sistema operativo.Copie tanto os arquivos binários executáveis como os arquivos suplementares da biblioteca. Assim sendo, seleccione para testes de programação um dos seguintes sistemas operativos:

- MS-Windows 32 bits:
  lua5_1_4_Win32_bin.zip
  lua5_1_4_Win32_dll8_lib.zip.
- MS-Windows 64 bits:
  lua5_1_4_Win64_bin.zip
  lua5_1_4_Win64_dll8_lib.zip
- Mac OS X Leopard:
  lua5_1_4_Darwin94x86_bin.tar.gz
  lua5_1_4_Darwin94x86_lib.tar.gz
- Linux 32 bits:
  lua5_1_4_Linux26g4_bin.tar.gz
  lua5_1_4_Linux26g4_lib.tar.gz
- Linux 64 bits:
  lua5_1_4_Linux26g4_64_bin.tar.gz
  lua5_1_4_Linux26g4_64_lib.tar.gz

Instalação da Linguagem Lua 5.1

Os procedimentos de instalação descritos em seguida são o mais simples possível e garantem o mínimo necessário para

o funcionamento da linguagem a partir da linha de comandos.
Caso tenha obtido a cópia para o sistema operativo MS-Windows 32 bits (ou 64 bits) crie uma pasta a partir da raiz do disco rígido com nome "lua5", descompacte os conteúdos dos dois arquivos para a pasta "lua5" e para fazer uso do interpretador Lua vá até a pasta "lua5" por meio da linha de Prompt do MS-DOS e digite no prompt a instrução:

```
lua5.1 nome_do_programa[.extensão]
```

Caso tenha obtido a cópia para o sistema operativo MacOS X Leopard ou Linux de 32 bits (ou de 64 bits) crie a partir da pasta do utilizador a pasta "lua5", descompacte os conteúdos dos dois arquivos na pasta "lua5" e para fazer uso do interpretador Lua vá até a pasta "lua5" por meio da janela Terminal digite no prompt a instrução:

```
./lua5.1 nome_do_programa[.extensão]
```

Variáveis e Constantes

Para usar um determinado dado é necessário que este dado esteja associado a uma variável, sendo esta variável a região de memória principal de um computador, previamente identificada por um rótulo, que tem por finalidade armazenar um determinado valor (dado) por um determinado espaço de tempo. Uma variável armazena apenas um valor de cada vez. É considerado valor um conteúdo numérico, alfabético, alfanumérico ou lógico.
O nome (também denominado rótulo) de uma variável é utilizado para sua identificação e posterior uso dentro de um programa, portanto é necessário estabelecer algumas regras para a definição do nome de uma variável:

- Nomes de variável podem ter um ou mais caracteres;
- O primeiro carácter do nome de uma variável não pode ser, em hipótese alguma, um número; deve ser sempre um carácter alfabético;
- O nome de uma variável não pode conter espaços em branco;
- Não pode ser um nome de uma variável uma palavra reservada a uma instrução ou identificador da linguagem Lua;
- Não podem ser utilizados outros caracteres no nome de uma variável, a não ser que sejam letras e números, com excepção do carácter underline "_", que pode ser usado para simular a separação de palavras compostas, como: NOME_ALUNO.

Deve-se ainda considerar que dentro de um programa uma variável pode exercer dois papéis. O papel de acção, quando é modificada ao longo do programa para apresentar um determinado resultado, e o segundo papel de controle, em que a variável será "vigiada" e controlada durante a execução de um programa (tipo de variável que será estudado nos capítulos que abordam a tomada de decisões e o uso de laços de repetição).
A linguagem de programação Lua permite o uso de variáveis sob dois propósitos, sendo o contexto local e o contexto global. Este tema será explorado com mais profundidade noutro artigo, sendo que neste interessa apenas o uso de variáveis sob a óptica do contexto global, que é a forma padrão de trabalho no uso de variáveis.

Os dados de uma linguagem de programação são representados por elementos a serem processadas por um computador electrónico. A linguagem de programação Lua fornece ao programador um conjunto de tipos de dados predefinidos e dinâmicos que são: nil (para a definição de um valor indefinido), number (para a definição de valores numéricos em notação científica, real ou inteiro), string (para a definição de valores alfanuméricos delimitados entre aspas ou entre apóstrofos e o uso de delimitadores de escape), function (para a definição de valor do tipo função), userdata (para a definição de valores do tipo apontador quando se associa a linguagem LUA com a linguagem C) e table (para a definição de tabelas de valores).
O tipo number é usado no sentido de representar valores numéricos, independentemente dos valores numéricos serem do tipo inteiro ou real. Por exemplo, o valor 5 pode ser representado em lua como:

```
A = 5
A = 5.0
A = 0.5e1
A = 50e-1
```

O tipo string é usado no sentido de representar uma cadeia de caracteres delimitada por aspas ou por apóstrofos que esteja definida em apenas uma linha de código ou então por meio de parênteses rectos duplos onde a sequência de caracteres poderá estar escrita em mais de uma linha de código. Dentro da fronteira de uma sequência de caracteres pode-se fazer uso das sequências de especiais:

\n	new line (nova linha)
\t	tab (tabulação horizontal)
\r	carriage return (retorno de carro – <Enter>)
\v	vertical tab (tabulação vertical)
\f	form feed
\xxx	carácter com código decimal xxx
\a	bell (campainha)
\b	backspace (retorno de espaço)
\"	aspas (")
\'	apóstrofo (')
\\	barra invertida (\)

A aplicação dos caracteres especiais dentro de uma sequência de caracteres (string) possibilita o uso de alguns efeitos dentro da sequência de caracteres em uso. Por exemplo:

```
TEXTO1 = "Utilize a tecla '<Enter>'"
TEXTO2 = 'Utilize a tecla \'<Enter>\''
```

Observe que a variável TEXTO1 está a ser atribuída com o valor da sequência de caracteres "Utilize a tecla '<Enter>'". Note que o uso de aspas simples dentro da sequência de caracteres ocorre de forma simples por estarem os símbolos de apóstrofos utilizados dentro da sequência de caracteres delimitada entre aspas inglesas ("). No entanto a variável TEXTO2 está a ser atribuída com o valor da sequência de caracteres 'Utilize a tecla \'<Enter>\''. Neste caso para poder fazer uso de aspas simples dentro da sequência de caracteres foi necessário o uso dos caracteres especiais \' entre a palavra <Enter> para que o carácter apóstrofo seja adequadamente apresentado.
Uma sequência de caracteres pode ser delimitada também entre parêntesis duplos ([[texto]]). Assim sendo, pode-se definir uma sequência de caracteres entre parêntesis duplos como:

```
TEXTO3 = [[Utilize a tecla '<Enter>']]
```

Ou pode-se ainda definir uma sequência de caracteres com parêntesis duplos como:

```
TEXTO3 = [[Utilize
a tecla
'<Enter>']]
```

Neste caso a mensagem será escrita em três linhas de texto exactamente como está definida na sequência de caracteres.
O tipo function é usado no sentido de representar valores que podem ser atribuídos a uma determinada variável que sejam passados como parâmetros. Como exemplo básico de uso deste tipo de dados considere o seguinte:

```
function nome_funçao(parametro)
  acçao
end

variável = nome_funçao(parametro)
```

O tipo userdata é usado no sentido de permitir que ocorra o armazenamento dentro de uma variável Lua de um apontador definido num código de programa escrito em C. Assim sendo, userdata corresponde ao tipo void* da linguagem de programação C, podendo ser utilizado apenas para comparação de igualdade ou atribuição entre valores do mesmo tipo tanto utilizados em C como utilizadores em Lua. O tipo userdata é útil em situações que há a necessidade de se fazer a ligação de código de programa entre a linguagem de programação Lua e a linguagem de programação C (Lua-C). No entanto, este tipo de dados não podem ser utilizados em programas escritos apenas com código de linguagem de programação Lua.
O tipo table é usado para a definição de variáveis compostas que permitem o uso de elementos indexados em matrizes de uma ou mais dimensões.
Os tipos de dados existentes na linguagem de programação lua não necessitam de ser explicitamente definidos para recorrer a sua utilização (por serem tipos dinâmicos) como ocorre, por exemplo, nas linguagens de programação Pascal, C, C++ entre outras.
Além da definição e uso de variáveis para a elaboração de programas, há também a necessidade de recorrer a constantes. Uma constante está normalmente associada à definição de valores pré-existentes, como valores que dão peso e equilíbrio em fórmulas e expressões matemáticas. Constante será a definição de um valor que ficará inalterado durante a execução de um programa.

Operadores e Expressões Aritméticas

Os operadores aritméticos são ferramentas responsáveis por permitir a elaboração e execução de cálculos matemáticos numa determinada linguagem de programação. Para se utilizar destes operadores é necessário fazer uso de variáveis e constantes.
Os operadores aritméticos podem ser classificados em duas categorias: binários ou unários. São binários quando actuam em operações de exponenciação, radiciação, multiplicação, divisão, adição e subtracção. São unários quando actuam na inversão da forma como um valor numérico será interpretado, atribuindo a este valor numérico o sinal positivo ou negativo.
A tabela seguinte apresenta um resumo dos operadores aritméticos utilizados e quais as operações matemáticas que podem ser executadas na linguagem de programação LUA:

Operador	Operação	Tipo	Resultado
=	Atribui um valor a uma variável	Unário	Atribuição
+	Manutenção de sinal	Unário	Positivo
-	Inversão de sinal	Unário	Negativo
^	Exponenciação	Binário	Number
sqrt(x)	Raiz quadrada	Unário	Number
x ^ (1/n))	Raiz de índice qualquer:	Binário	Number

É muito comum para um computador utilizar expressões aritméticas, uma vez que, na sua maioria, o trabalho computacional utiliza um grande número de cálculos. Essas expressões são definidas pelo relacionamento existente entre variáveis e constantes numéricas, com a utilização dos operadores aritméticos.

Atribuições e Coerções

As operações de atribuições estão relacionadas ao fato de transferir explicitamente um valor a uma variável, podendo está atribuição ocorrer de duas formas: simples e múltipla.
As atribuições simples caracterizam-se por serem definidas de acordo com o formato VARIÁVEL = VALOR, onde VALOR está a ser atribuído a VARIÁVEL.
Já as atribuições múltiplas caracterizam-se por serem definidas de acordo com o formato VARIÁVEL1, VARIÁVEL2, VARIÁVELN = VALOR1, VALOR2, VALORN; onde VALOR1 está a ser atribuído a VARIÁVEL1, VALOR2 atribuído a VARIÁVEL2 e VALORN atribuído a VARIÁVELN.
Caso ocorra a definição de um número de variáveis à esquerda do sinal de atribuição diferente do número de valores à direita do sinal de atribuição ocorrerá automaticamente uma de duas possibilidades, ou ocorrerá o preenchimento de valores das variáveis com o valor null, quando a lista de variáveis for maior que a lista de valores, ou os valores serão descartados, quando a lista de variáveis for menor que a lista de valores.
O recurso de atribuições múltiplas permite fazer a troca de valores entre variáveis em apenas numa linha de código. Normalmente para realizar esta acção, utiliza-se três linhas de código. Observe a seguir um comparativo de troca de valores efectuados nas linguagens PASCAL, C, BASIC e Lua.

PASCAL	C	BASIC	Lua
X := A; A := B; B := X;	X = A; A = B; B = X;	X = A A = B B = C	A, B = B, A

Uma ocorrência característica da linguagem de programa Lua é o efeito de coerção que ocorre quando uma determinada operação de atribuição é realizada. Lua é uma linguagem que define automaticamente para uma variável em uso o tipo de dados que vão ser associado. Não é necessário definir previamente o tipo da variável em uso.

O efeito de coerção (conversão automática) tenta aplicar sobre uma operação aritmética a definição de dados do tipo numérico, mesmo quando está em uso um dado do tipo sequência de caracteres (string), que seja é claro, a representação de uma sequência numérica. Por exemplo:

```
A = 20
B = "30"
C = A + B
```

Observe que no caso anterior a variável C possuiria o valor numérico 50, sendo este a soma do valor numérico 20 da variável A com o valor carácter "30" da variável B convertido automaticamente para o seu valor numérico equivalente 30.
No caso anterior ocorreu o cálculo de C do conteúdo da variável B apesar de ser uma string, pois o mesmo representava um valor numérico correspondente a quantidade 30. É claro que quando não for possível estabelecer uma operação de coerção a linguagem Lua reportará um erro de execução.
Outra possibilidade de uso de coerção é quando se deseja concatenar um valor numérico a um valor string definido. Por exemplo:

```
A = 20
B = "Resultado = "
C = A .. B
```

Observe que no caso anterior a variável C possuiria o valor concatenado: Resultado = 20, onde o conteúdo 20 da variável A será tratado como se fosse "20" e será concatenado por meio do operador aritmético .. com o conteúdo "Resultado = " da variável B.

Entrada Processamento e Saída

A acção de entrada é responsável por permitir a um determinado utilizador de um programa fornecer os dados que serão armazenados na memória (das variáveis) para posterior uso na fase de processamento.
A acção de processamento pode ocorrer sobre dois aspectos básicos: matemático ou lógico. O processamento matemático ocorre quando se faz uso de operações matemáticas e o processamento lógico ocorre quando se utiliza controlos em tomadas de decisão, execução de acções repetitivas e em qualquer operação que não envolva cálculos matemáticos.
A acção de saída ocorre normalmente após a conclusão de uma acção de processamento ou de uma acção de entrada, permitindo assim apresentar ao utilizador um determinado valor como resposta de uma acção anterior realizada.
Uma entrada e uma saída podem ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada pode ser feita via teclado, modem, leitores ópticos, disco, scanners entre outros. Uma saída pode ser feita num monitor de vídeo, impressora, disco, entre outros periféricos. Devido a esta grande variedade, os programas desta obra utilizarão as instruções io.read() (para a entrada

de dados via teclado) e io.write() ou print() (para a saída de dados em monitor de vídeo - ecrã).
Para a saída de dados a linguagem Lua disponibiliza a possibilidade de uso de duas instruções, sendo io.wirite() utilizado quando se deseja em média manter o cursor posicionado na mesma linha (pode-se alterar o cursor de linha se utilizado a sequencia de \n) após a apresentação de um determinado conteúdo, ou print() quando se deseja que o cursor seja posicionado na linha seguinte após a apresentação do conteúdo.
Nas instruções io.read() e io.write() têm na sua composição a definição de dois seguimentos separados por ponto. Note que o primeiro seguimento de cada instrução é formado por io (input/output) que informa que os comandos read e write pertencem a uma biblioteca denominada io (biblioteca de entrada e saída).
Para colocar em prática o exposto até este momento, considere o desenvolvimento de um programa que efectue a leitura de dois valores numéricos, faça em seguida a operação de adição (processamento matemático) dos dois valores lidos e apresente como saída o resultado obtido da adição processada. Considere o código:

```
-- inicio do programa ADICIONA NUMEROS
  A = io.read("*number")
  B = io.read("*number")
  X = A + B
  io.write(X, "\n")
-- fim do programa ADICIONA NUMEROS
```

Em seguida escreva o código do programa anterior num editor de texto da sua preferência, gravando-o com o nome progro1.lua.
Observe que o código do programa progro1.lua é formado por seis linhas. Cada linha representa a definição de uma instrução de execução de uma determinada acção computacional.
A primeira e a última linha de código, indicadas pelo uso dos caracteres traços duplos, são usadas no sentido de definir linhas de comentários. O uso de linhas de comentários permite a possibilidade de se fazer a documentação de informações consideradas importantes. No exemplo apresentado as linhas de comentários indicam o início e o fim do programa ADICIONA NUMEROS.
A segunda e terceira linhas estão a ser utilizadas no sentido de efectivarem as instruções de entrada dos valores numéricos e de transferirem estes valores respectivamente para as variáveis A e B. Observe que para efectivar a entrada de dados está a ser utilizado o comando io.read() com o parâmetro *number entre aspas informando que o conteúdo a ser armazenado na variável, que é um dado de tipo numérico e que poderá ser do tipo inteiro ou real. Se o parâmetro *number for omitido, o interpretador entenderá que a entrada de dados é um valor do tipo alfanumérico. Além do parâmetro *number pode-se também fazer uso dos parâmetros *all para fazer a leitura de um arquivo inteiro em uso ou *line para fazer a leitura da próxima linha de um arquivo em uso. Nesta etapa de aprendizagem será concentrado o estudo apenas sobre o parâmetro *number.
A quarta linha de código estabelece o uso de um processamento matemático, cuja acção efectua a adição dos valores armazenados nas variáveis A e B e efectua a atribuição do resultado da adição dentro da variável X.
A quinta linha do programa faz a saída do conteúdo armazenado na variável X por meio da acção anterior de processamento. Note que além da indicação da variável X, há também a definição da sequência de especial \n no sentido de avançar uma linha em branco após a apresentação da saída do programa.
Observe no programa progro1.lua o uso dos conceitos de entrada, processamento e saída para a acção pretendida pelo programa. As figuras 2.1 e 2.2 mostram respectivamente a listagem do código e a execução do programa após o uso da chamada:

```
lua5.1 progr01.lua
```

```
./lua5.1 progr01.lua
```

Perceba que na execução do programa é apresentado um cursor piscando no ecrã. Forneça o primeiro valor numérico. Introduza o valor 5 e pressione a tecla <Enter>.Na sequência entre um segundo valor, introduza o valor 4.5 e pressione a tecla <Enter>. Perceba em seguida a apresentação do resultado será 9.5.
Apesar do programa progro1.lua apresentar correctamente o valor da soma, não interage adequadamente com o seu utilizador. Para melhorar a legibilidade do programa com o seu utilizador é necessário fazer algumas alterações. Observe as linhas em negrito inseridas no código:

```
-- inicio do programa ADICIONA NUMEROS

  io.write("Entre o 1o. valor: ")
  A = io.read("*number")
  io.write("Entre o 2o. valor: ")
  B = io.read("*number")
  X = A + B
  io.write("O resultado da soma
equivale a: ", X, "\n")

-- fim do programa ADICIONA NUMEROS
```

Grave a nova versão do programa com o nome progro2.lua e em seguida execute na linha de comando (para Windows e Linux respectivamente):

```
lua5.1 progr02.lua
```

```
./lua5.1 progr02.lua
```

Com o uso das mensagens de entrada e saída a acção do programa fica mais clara para o utilizador.
No sentido de exemplificar o uso do comando print(), proceda à alteração indicada em negrito do código seguinte.
Após a alteração proposta, grave a nova versão do programa como o nome progro3.lua.

```
-- inicio do programa ADICIONA NUMEROS

  io.write("Entre o 1o. valor: ")
  A = io.read("*number")
  io.write("Entre o 2o. valor: ")
  B = io.read("*number")
  X = A + B
  print("O resultado da soma equivale
a: ", X)

-- fim do programa ADICIONA NUMEROS
```

Após a execução do programa na linha de comando perceba o efeito de apresentação da mensagem de saída do programa. Note que a instrução com uso do comando print() efectua o salto automático da linha e apresenta antes do valor de saída alguns espaços em branco.
A seguir será desenvolvido um programa que calcula o salário líquido de um profissional que trabalha por hora. Para montar o programa, você deve possuir alguns dados, tais como: valor da hora de trabalho, número de horas de trabalho no mês e a percentagem de desconto da assistência médica. O programa em questão deve apresentar o valor do salário bruto, o valor descontado e o valor do salário líquido. Assim sendo, observe o código do programa progro3.lua.

```
-- inicio do programa SALARIO
  io.write("Quantas horas de trabalho?
")
  HT = io.read("*number")
  io.write("Qual o valor da hora? ")
  VH = io.read("*number")
  io.write("Qual o percentual de
desconto? ")
  PD = io.read("*number")
  SB = HT * VH
  TD = (PD/100) * SB
  SL = SB - TD
  io.write("Salario bruto ...: ", SB,
"\n")
  io.write("Desconto ........: ", TD,
"\n")
  io.write("Salario liquido .: ", SL,
"\n")
-- fim do programa SALARIO
```

Grave o programa com o nome progro3.lua e em seguida execute-o.
Para tornar a visualização da saída dos dados numéricos mais adequada, deverão ser utilizados junto as instruções de saída io.write() a função string.format() com a formatação dos valores numéricos. Assim sendo, será utilizado o formato numérico de proporção 7:2:

7						
5					2	
9	9	9	9	.	9	9
Expoente				Ponto	Mantissa	

Observe no quadro anterior o uso de sete casas de posicionamento numérico, das sete casas duas casas serão usadas para a apresentação da mantissa do valor, uma será u para a apresentação do ponto de separação decimal, sobrando para o expoente total de quatro casas. Desta forma, os valores numéricos com tamanho até 9999.99 serão alinhados da direita para à esquerda até este limite os valores das casas de milhar, centena, dezena, unidade e posições decimais.
A seguir efectue as alterações indicadas em negrito do código do programa progro3.lua e grave as modificações num ficheiro de nome progro4.lua.

```
-- inicio do programa SALARIO

  io.write("Quantas horas de trabalho?
")
  HT = io.read("*number")
  io.write("Qual o valor da hora? ")
  VH = io.read("*number")
  io.write("Qual o percentual de
desconto? ")
  PD = io.read("*number")
  SB = HT * VH
  TD = (PD/100) * SB
  SL = SB - TD
  io.write("Salario bruto ...: ",
string.format("%7.2f", SB), "\n")
  io.write("Desconto ........: ",
string.format("%7.2f", TD), "\n")
  io.write("Salario liquido .: ",
string.format("%7.2f", SL), "\n")

-- fim do programa SALARIO
```

O primeiro valor, no caso 7, corresponde ao tamanho total de posições de um número do tipo ponto flutuante, e o segundo valor corresponde ao número de casas decimais após o ponto (mantissa) mais o ponto que será subtraído do

tamanho total.
Observe que para formatar a saída numérica com ponto flutuante foi utilizado o formato %TOTAL.MATISSAf, onde TOTAL é a definição do total de posições a ser utilizada (expoente + ponto + mantissa), e MANTISSA é a definição da quantidade de casas decimais a ser utilizada. O código f é usado para definir a formatação de valores em ponto flutuante. Caso queira formatar uma saída numérica para valores inteiros basta fazer uso do formato %TOTALd, onde TOTAL é a definição do tamanho numérico a ser usado e o código d é usado para definir a apresentação de valores inteiros.

Além das formatações apresentadas é possível fazer a definição de outras formas. Assim sendo, pode-se fazer de alguns formatadores como: %s string; %q string com delimitadores, num formato que possa ser lido pela linguagem Lua; %c carácter; %d inteiro com sinal; %i igual a %d; %u inteiro sem sinal; %o inteiro octal; %x hexadecimal usando letras minúsculas (abcdef); %X hexadecimal usando letras maiúsculas (ABCDEF); %f real no formato [-]ddd.ddd e %e real no formato [-]d.ddd.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem 23 anos de experiência em ensino e desenvolvimento de programação de software. É professor da rede federal de ensino no Brasil, no Centro Federal de Educação Tecnológica de São Paulo. É também autor, possuindo na sua carreira mais de cinquenta obras publicadas.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

Templates T4

Introdução

Uma das áreas que me tem interessado e motivado mais nos últimos tempos, é a de geração automática de código, especialmente aplicado a frameworks aplicacionais. Depois de passar horas a escrever código à "moda antiga", a escrever o mesmo tipo de código constantemente, a ideia de poder gerar automaticamente o código é sem dúvida entusiasmante. E torna-se ainda mais interessante se poder gerar uma estrutura de domínio completa a partir de um pequeno bloco de informação - uma framework que reduz o tempo de desenvolver código tediante. Não me refiro a código que uma melhor estrutura de objectos e hierarquias de herança resolva: refiro-me a objectos que apenas varia nos nomes da classe ou dos campos ou tipo de dados, e que Generics (ou equivalente) não resolvem completamente.

Há várias formas de poder gerar o código de forma automática. É necessário pelo menos um conjunto de metadados (que descreve os objectos do nosso domínio) e um processador para transformar essa informação. O conjunto de metadados pode ser XML, estrutura ou dados de bases de dados, ou um simples ficheiro texto devidamente formatado. É conveniente que os dados sejam estruturados para maximizar as potencialidades. Se o processador se basear em templates, basta aplicar um ou mais templates para transformar os metadados em ficheiros com código funcional. E podemos evoluir as saídas de forma dinâmica. Qualquer alteração a um template altera todos os objectos criados: qualquer bug genérico introduzido num template é corrigido em todos os objectos criados.

E o melhor de tudo, o caro leitor carrega num botão e tem milhares e milhares de linhas de código escrito automaticamente!!! Deve ser claro que não é a "solução maravilha" para todas as situações, nem a "bala prateada", mas resolve bem muitas situações frequentes, e ficamos com tempo para fazer o que realmente interessa que é atacar os problemas e algoritmos mais complexos.

Text Template Transformation Toolkit (ou T4 para os amigos)

Uma das muitas formas de gerar código é a utilização de templates T4 (Text Template Transformation Toolkit). O T4 é um motor de processamento de templates integrado directamente no Visual Studio 2008 (para o 2005 é necessário instalar o DSL Toolkit), e permite adicionar e processar automaticamente os templates num projecto de biblioteca de classes do VS. É tão simples como criar um ficheiro com a extensão ".tt" no projecto que representa um template específico. Em qualquer momento em que o ficheiro de template é armazenado, o template é processado e o ficheiro resultante é criado (ou actualizado). Por defeito, este ficheiro é da extensão do ambiente (.cs se desenvolveres em C#, por exemplo) e incluído imediatamente no projecto em que está inserido, entrando no processo de compilação. No entanto, é possível definir que o ficheiro resultante tenha outro tipo de extensão, como .html, .sql, .aspx, .ascx, .xml, etc.

Sempre que é criado um ficheiro template, é automaticamente criado um ficheiro vazio com o mesmo nome e a extensão de resultado por defeito (normalmente .cs ou .vb, dependendo das definições do ambiente). O template é, essencialmente, um ficheiro com o script de transformação. Cada template resulta num novo ficheiro, e o conteúdo é o resultado do script de transformação. O resultado pode ser qualquer tipo de ficheiro textual - código C# ou VB, aspx, html, javascript, xml, ou texto. A conversão é precedida de uma compilação do script para prevenir erros.

O processo de transformação dos templates é relativamente complexa, especialmente se construirmos o nosso próprio processador de templates, mas é o que o torna poderoso. De uma forma sucinta, o nosso template (.tt) é transformado numa classe com o nome do ficheiro do template, num espaço de nomes temporário usado pelo Visual Studio. O texto e instruções que temos no template é convertido em instruções (essencialmente de Write() e WriteLine()) de um método da classe criada chamado TransformText(). Se tudo correr bem neste processo, esta classe é compilada e instanciado pelo motor de processamento, e o resultado do TransformText() é escrito para o ficheiro de saída.

Vamos então analisar a sintaxe de um template T4 e ver o que pode ser feito com o mesmo.

Estrutura de um template

Vamos entrar já na criação de um template simples para conhecer a sintaxe. Primeiro passo é adicionar um template ao projecto: basta adicionar um ficheiro (de texto por exemplo) com a extensão .tt. Para este caso, o HelloWorld.tt

parece-se apropriado. Note que, ao ser adicionado o ficheiro ao projecto, um ficheiro novo, HelloWorld.cs, é adicionado. É o template a entrar em acção. Obviamente, porque não fizemos mais nada, o ficheiro de resultado estará vazio.

Sempre que gravamos um ficheiro .tt, o Visual Studio transforma-o automaticamente. Se tivermos vários templates e quisermos transformar todos, de uma só passagem, devemos pressionar o botão no topo da solução "Transform All Templates".

Dica: Se tiver o T4 editor (menciono-o mais à frente) há itens novos adicionados à lista de itens que podem ser usados num projecto. Um deles é o T4 template que cria um ficheiro com a extensão .t4, que não é um template transformado imediatamente. No entanto, se o escolher e alterar a extensão para .tt, as directivas iniciais são introduzidas automaticamente.

Sintaxe

Os ficheiros t4 devem iniciar com directivas próprias da linguagem e que serão utilizadas para preparar o motor de processamento.

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System" #>
```

Qualquer bloco executável num template começa com "<#" e termina com o "#>", similar ao "<% %>" do ASP.NET. As directivas tem o @ extra na abertura do bloco. Há 6 directivas pré-definidas:

Directiva	Utilização
<#@ Template ... #>	Directiva principal que define a linguagem usada no template. Aceita C#, C#v3.5, VB ou VBv3.5.
<#@ Import ... #>	Permite importar namespaces para utilizar as classes desses espaços no código do template. O System é a base, e praticamente sempre presente.
<#@ Include ... #>	Permite importar ficheiros para dentro do template que podem ter blocos de código ou funções reaproveitáveis.
<#@ Assembly ... #>	Permite importar assemblagens a partir do nome ou caminho físico do ficheiro, equivalente a incluir a referência da assemblagem num projecto .NET.
<#@ Output ... #>	Permite alterar as definições do ficheiro resultante, nomeadamente a extensão do ficheiro e a codificação. Extenções como o .cs e .vb entram no ciclo de compilação do projecto em que está inserido.
<#@ Property ...#>	Define uma propriedade a usar durante o processamento da transformação.

Com isto, podemos ver que o código que vou utilizar no script é o C# na versão 3.5 do .NET, e que é importado o namespace "System" para o template, para poder usar os objectos de base do .NET. Por omissão, o ficheiro gerado é da extensão da linguagem preferida no IDE (no meu caso .cs). Para altera, por exemplo, para .html, basta adicionar a directiva de output:

```
<#@ Output Extension=".html" #>
```

Vamos então adicionar a informação que queremos incluir no ficheiro resultante. Temos dois tipos de de blocos informação que podemos incluir - texto que é escrito directamente, ou blocos de código que serão processados e que resultam em texto. Por exemplo, se no meu template eu incluir:

HeloWorld.tt

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System" #>
<#@ Output Extension=".html" #>
<html>
   <body>
      Olá Mundo!
   <body>
</html>
```

O ficheiro resultante terá escrito:

```
<html>
   <body>
      Olá Mundo!
   <body>
</html>
```

Mas o poder dos templates surge quando incluímos blocos de código e dados dinâmicos, como por exemplo:

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System" #>
<#@ Output Extension=".html" #>
<html>
   <body>
      Olá Mundo! Fui criado às <#=
DateTime.Now.ToString("HH:mm") #>
   <body>
</html>
```

que resulta em:

```
<html>
   <body>
      Olá Mundo! Fui criado às 20:15
   <body>
</html>
```

Temos assim um exemplo de um timestamp num ficheiro gerado automaticamente. Este exemplo apresenta um novo bloco de código. Há 4 tipos de blocos:

Tipo	Descrição
<#	Bloco de código na linguagem .NET definida e que é inserida directamente no método TransformText() da classe de transformação. Podemos englobar sequências de instruções, condições e ciclos de controlo dentro de blocos deste género, e ainda forçar a escrita de informação no ficheiro de saída com os métodos Write() e WriteLine().
<#=	Escreve o resultado de uma expressão, similar a um databind no ASP.NET
<#+	Adiciona "class features" ao código, permitindo criar métodos e classes reutilizáveis nos templates
<#@	Directivas que apresentam instruções de preparação do motor de processamento

Um gerador de código simples

Vimos as bases. A nível de sintaxe não há muito mais para ver. Resta ver o potencial de transformação do template em código. Automatizar o processo é essencial, especialmente em estruturas complexas. Depois de gerar algumas aplicações (ou mesmo no decorrer de um projecto concreto), há sempre padrões de construção que encontramos e que tentamos tornar mais coerente efectuando um refactoring para maximizar o reaproveitamento de código e a organização dos objectos segundo padrões de POO.

Mesmo assim, há estruturas que ultrapassam a estrutura de objectos, como nomes, tipos de dados, descrições, mapeamento de campos, etc. Um domínio especifico pode ter uma forma coerente de construir objectos com estes parâmetros, e um gerador de código pode ser usado para criar o código do domínio automaticamente. Imagina uma aplicação de n-camadas, com UI, Camada de negócios, Camada de acesso a dados, e a própria estrutura de armazenamento de dados. Tem regras próprias, que em alguns casos podem ser generalizadas, mas as características dos objectos do domínio não são genéricas. Com o gerador de código podemos cria-los e a funcionalidade associada automaticamente!

Vou então pegar neste exemplo, simplificado, para demonstrar o potencial do gerador. Vou considerar a seguinte estrutura:

- A minha base de dados é um conjunto de tabelas com colunas simples (vou omitir relações pela simplicidade e podermos concentrar no T4)

- A minha aplicação tem como Business Objects (BO) classes com propriedades, que reflectem as tabelas da base de dados.

- Para aceder à base de dados, uso um conjunto de classes que vou denominar DAL (Data Access Layer), que executam comandos CRUD sobre a BD, e preenchem o objecto.

- A UI utiliza os BO para apresentar os dados e os CRUD da DAL para interagir com a BD (estou a omitir uma camada de negócio para simplificar).

Com o gerador de código vou tentar criar o SQL de criação da base de dados, alguns métodos da DAL, e as classes BO.

Definir o suporte da metainformação

Para gerar o código, preciso de uma estrutura de metadados que descreva o meu domínio. Lembrando que os BO no nosso caso reflectem directamente as tabelas da base, podemos considerar que as classes equivalem ás tabelas e as propriedades às colunas. Uma classe é essencialmente uma lista de propriedades tal como uma tabela é uma lista de colunas.

Posto isto, sei que:

- classes e tabelas tem o mesmo nome
- propriedades reflectem colunas
    - propriedades e colunas tem nomes
    - uma coluna pode ser uma chave primaria
    - uma propriedade pode ter um valor por defeito
    - o tipo de propriedade e o tipo de coluna devem ser equivalentes

A simplicidade permite que eu defina esta estrutura num objecto que posso usar em código.

Vou então definir isto num ficheiro .t4 que adiciono ao meu projecto. Podia efectua-lo num ficheiro .tt, mas este iria gerar um ficheiro vazio extra desnecessário:

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System"  #>
<#@ Import
Namespace="System.Collections.Generic"
#>
<#+

public class MinhaPropriedade{
    public string Nome {get; set;}
    public bool IsPK { get; set; }
    public string TipoCS { get; set; }
    public string TipoDB { get; set; }
    public string ValorDefeito { get;
set; }
    public MinhaPropriedade(string
nome, bool isPK, string tipocs, string
tipodb, string valordefeito){
        this.Nome = nome;
        this.IsPK = isPK;
        this.TipoCS = tipocs;
        this.TipoDB = tipodb;
        this.ValorDefeito =
valordefeito;
    }
}

public class MinhaClasse{
    public string Nome { get; set; }
    public List<MinhaPropriedade>
Propriedades { get; set; }
    public MinhaClasse(string nome){
        this.Nome = nome;
        this.Propriedades = new
List<MinhaPropriedade>();
    }
}

#>
```

O ficheiro começa com as directivas do template e imports necessários. Depois temos o inicio de bloco "<#+" com que podemos definir estruturas de classes. Como vamos incluir este ficheiro noutros ficheiros de templates, as classes aqui definidas serão disponibilizadas nesses ficheiros. Basta incluir o a referência a este ficheiro (como veremos a seguir) para poder usar as classes aqui definidas. O código apresentado no bloco é de simples definição de classe e propriedades.

Assim sendo tenho duas classes disponíveis: MinhaClasse em que defino uma class/tabela e MinhaPropriedade que define propriedades/colunas. Vou querer criar mais uma classe que vai servir para carregar um objecto para gerar código a partir dele. Em casos reais, as estruturas serão mais complexas e portanto estruturas XML ou DSLs completas (que o VS consegue-se criar usando os SDKs disponíveis) ou bases de dados serão usadas para suportar os metadados. Para este caso vamos simplificar e carregar os dados directamente.

Loader.t4

```
#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System" #>
<# //@ Include File="Classes.t4" #>
<#+

        public class Loader
        {
            public static MinhaClasse
CarregarDefinicao()
            {
                MinhaClasse minhaClasse
= new MinhaClasse("ObjectoSimples");

minhaClasse.Propriedades.Add(new
MinhaPropriedade("id", true, "int",
"int", null));

minhaClasse.Propriedades.Add(new
MinhaPropriedade("nome", false,
"string", "varchar(100)",
"String.Empty"));

minhaClasse.Propriedades.Add(new
MinhaPropriedade("funcao", false,
"string", "text", "String.Empty"));

                return minhaClasse;
            }
        }

#>
```

Nota: Novamente, há as directivas bases, e uma que permite incluir o "Classes.t4". Durante a escrita da classe, é útil incluir esta directiva para facilitar a escrita, mas posteriormente é necessário comentar para evitar qualquer duplicação das classes de Classe.t4 nos templates. Também de notar, o Include refere um caminho relativo ao ficheiro actual.

Neste ficheiros definimos uma classe Loader com um único método estático que retorna os dados da classe que vamos usar. Novamente, a classe está definida entre marcadores do tipo "<#+ #>" Numa DSL (Domain Specific Language) criada no VS, a definição da mesma cria um processador que carrega a estrutura a partir de XML. Podíamos também definir uma estrutura do género em XML ou até em ficheiro texto bem definido, e métodos específicos para o carregar.
Definir os objectos

Vamos gerar então um objecto de negócio BO. No exemplo vou apenas mostrar uma única classe, e portanto o ficheiro de saída será directamente incluído no projecto. Vamos então criar o ObjectoSimples.tt:

ObjectoSimples.tt

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System"  #>
<#@ Import
Namespace="System.Collections.Generic"
#>
<#@ Output Extension=".cs" #>
<#@ Include File="Classes.t4"  #>
<#@ Include File="Loader.t4" #>
<#
      MinhaClasse minhaClasse =
Loader.CarregarDefinicao();
#>
using System;

public class <#= minhaClasse.Nome #>
{
<#
    foreach(MinhaPropriedade
propriedade in
minhaClasse.Propriedades)
    {
#>
    private <#= propriedade.TipoCS #>
_<#= propriedade.Nome #><#=
propriedade.ValorDefeito != null ? " =
" + propriedade.ValorDefeito : "" #>;
    public <#= propriedade.TipoCS #>
<#= propriedade.Nome #>
    {
        get { return _<#=
propriedade.Nome #>; }
        set { _<#= propriedade.Nome #>
= value; }
    }

<#
    }
#>
}
```

Temos agora um ficheiro com as directivas necessárias. Para além do template e imports, temos os includes dos t4 que definimos anteriormente, e ainda indicamos explicitamente que o ficheiro gerado pelo template tem a extensão ".cs".

O próximo bloco é do tipo <# #> (sem qualquer carácter especial adicional). Este tipo de bloco é incluído directamnete no TransformText() da classe criada a partir do template. Neste caso é instanciado MinhaClasse e carregado os dados da classe através da chamada a Loader.CarregarDefinicao().

De seguida, é escrito texto directamente para o ficheiro de saída, que neste caso será o "using" e o inicio da declaração da classe. O nome da classe é incluído usando <#= minhaClasse.Nome #>, onde acedemos directamente à propriedade Nome da instancia minhaClasse que criamos. Após a declaração da classe, listamos as propriedades na mesma, usando um ciclo foreach num conjunto de blocos <# #>

```
using System;
public class ObjectoSimples
{
    private int _id;
    public int id
    {
        get { return _id; }
        set { _id = value; }
    }
    private string _nome =
String.Empty;
    public string nome
    {
        get { return _nome; }
        set { _nome = value; }
    }
    private string _funcao =
String.Empty;
    public string funcao
    {
        get { return _funcao; }
        set { _funcao = value; }
    }

}
```

O template gerou assim o código do objecto com os campos privados e as propriedades publicas, getters e setters incluídos. Podia ainda incluir atributos de serialização, comentários de descrição, etc. Quaisquer dados necessários devem ser colocados nos metadados e usados pelo template para incluír no código com a formatação desejada.

Definir script da base de dados

Vamos gerar SQL agora, para criar a tabela na base de dados:

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System"  #>
<#@ Import
Namespace="System.Collections.Generic"
#>
<#@ Output Extension=".sql" #>
<#@ Include File="Loader.t4" #>
<#@ Include File="Classes.t4" #>
<#
      MinhaClasse minhaClasse =
Loader.CarregarDefinicao();
#>
CREATE TABLE <#= minhaClasse.Nome #>
{
<#
      for (int i = 0; i <
minhaClasse.Propriedades.Count; i++ )
      {
          MinhaPropriedade propriedade
= minhaClasse.Propriedades[i];
#>
    <#= propriedade.Nome #> <#=
propriedade.TipoDB #> <#=
propriedade.IsPK ? "PRIMARY KEY" : ""
#><#=
i<minhaClasse.Propriedades.Count-1 ?
"," : "" #>
<#
      }
#>
}
```

que gera:

```
CREATE TABLE ObjectoSimples
{
    id int PRIMARY KEY,
    nome varchar(100) ,
    funcao text
}
```

Repara que, a partir da mesma definição, foi possível criar dois ficheiros de código de tipos diferentes, baseados maioritariamente em parâmetros comuns. Neste caso definimos a saída do tipo .sql. Um ficheiro SQL deste tipo teria o script de criação da base, incluindo relações a adicionar, sequências, stored procedures base, etc. O template que gerei foi direccionado para SQL server, mas podia perfeitamente direccionar a outro SGBD ou até criar mais um template dedicado a outra base de dados.
Definir o acesso a dados

Para a DAL, teremos essencialmente uma classe com um conjunto de métodos que implementa métodos CRUD comuns como obter um registo por um ID, uma lista por um conjunto de parâmetros, número de registos por parâmetros, inserção e actualização de registos, actualização determinado campo de um registo ou ainda apagar registos. Deve ainda ser capaz de mapear os dados dos campos da base de dados para as propriedades dos objectos respectivos. No exemplo que segue, demonstro uma versão simples de acesso a um registo e preenchimento de um objecto:

ObjectoSimplesDAL.tt

```
<#@ Template Language="C#v3.5" #>
<#@ Import Namespace="System"  #>
<#@ Import
Namespace="System.Collections.Generic"
#>
<#@ Output Extension=".cs" #>
<#@ Include File="Loader.t4" #>
<#@ Include File="Classes.t4" #>
<#
      MinhaClasse minhaClasse =
Loader.CarregarDefinicao();
#>
using System;
using ClassesGenericasDeAcessoADados;

public class <#= minhaClasse.Nome
#>DAL
{
    /// <summary>
    /// Retorna um objecto do tipo <#=
minhaClasse.Nome #> a partir da chave
    /// </summary>
<#
      MinhaPropriedade chave = null;
      foreach (MinhaPropriedade prop
in minhaClasse.Propriedades)
      {
          if (prop.IsPK)
          {
              chave = prop;
              break;
          }
```

```
        }
#>
    /// <param name="<#= chave.Nome
#>">ID a pesquisar</param>
    /// <returns></returns>
    public <#= minhaClasse.Nome #>
GetItem(<#= chave.TipoCS #> <#=
chave.Nome.ToLower() #>)
    {
        string sqlQuery = "SELECT *
FROM <#= minhaClasse.Nome #> WHERE <#=
chave.Nome #> = " + <#=
chave.Nome.ToLower() #>;

        try
        {
            using(DataAccessor
dataAccessor = new
DataAccessor(connectionString) )
                return (<#=
minhaClasse.Nome
#>)dataAccessor.GetDBObject(sqlQuery,
FillObject);
                }
                catch(Exception e)
                {
                //código de logging
                }
    }

        /// <summary>
        /// Preenche <#=
minhaClasse.Nome #> com os dados
provenientes da base
        /// </summary>
        /// <param
name="data">IDataReader com os dados
recebidos</param>
        /// <returns></returns>
    public static <#= minhaClasse.Nome
#> FillObject(IDataReader data)
    {
        <#= minhaClasse.Nome #> obj =
new <#= minhaClasse.Nome #>();

        using (DataParser dataParser =
new DataParser(data))
                {
<#
    foreach(MinhaPropriedade
propriedade in
minhaClasse.Propriedades)
    {
#>
            if (dataParser.IsValid<#=
propriedade.TipoCS #>("<#=
propriedade.Nome #>"))
                        obj.<#=
propriedade.Nome #> =
dataParser.DRGet<#= propriedade.TipoCS
#>("<#= propriedade.Nome #>");
<#
    }
#>
        }

        return obj;
    }
}
```

que resulta em:

```
using System;
using ClassesGenericasDeAcessoADados;

public class ObjectoSimplesDAL
{

        /// <summary>
    /// Retorna um objecto do tipo
ObjectoSimples a partir da chave
    /// </summary>
    /// <param name="id">ID a
pesquisar</param>
    /// <returns></returns>
    public ObjectoSimples GetItem(int
id)
    {
        string sqlQuery = "SELECT *
FROM ObjectoSimples WHERE id = " + id;

        try
        {
            using(DataAccessor
dataAccessor = new
DataAccessor(connectionString) )
                return
(ObjectoSimples)dataAccessor.GetDBObjec
t(sqlQuery, FillObject);
                }
                catch(Exception e)
                {
            //código de logging
                }
    }

    /// <summary>
        /// Preenche ObjectoSimples
com os dados provenientes da base
        /// </summary>
        /// <param
```

```
name="data">IDataReader com os dados
recebidos</param>
        /// <returns></returns>
    public static ObjectoSimples
FillObject(IDataReader data)
    {
        ObjectoSimples obj = new
ObjectoSimples();

        using (DataParser dataParser =
new DataParser(data))
                {
            if
(dataParser.IsValidint("id"))
                obj.id =
dataParser.DRGetint("id");
            if
(dataParser.IsValidstring("nome"))
                obj.nome =
dataParser.DRGetstring("nome");
            if
(dataParser.IsValidstring("funcao"))
                obj.funcao =
dataParser.DRGetstring("funcao");

        }

        return obj;
    }
}
```

Repare que foi possível uniformizar os comentários XML para documentação, que ficam simples de manter, e ainda uniformizar a estrutura de de acesso a dados. DataAccessor é um objecto com operações genericas (como o getDBObject) implementadas no namespace "ClassesGenericasDeAcessoADados". GetDBObject está intergrado nesse objecto, executa uma query na base, e devolve um objecto preenchido através do método delegado FillObject.

FillObject também é criado no template e integrado na classe da DAL. Essencialmente, utiliza uma classe DataParser com alguns métodos de validação e obtenção de dados, preenchendo as propriedade de ObjectoSimples com os dados das colunas respectivas.

Com estas classes já geradas, podíamos perfeitamente compilar os ficheiros num projecto de DLL e incluí-lo em aplicações que pretendemos construir para o domínio dos nossos objectos. Quaisquer alterações ao domínio, evolveria apenas regenerar os ficheiros e recompilar, substituindo a .dll original.

O Processo de Transformação

Até agora, vimos a sintaxe dos templates, como podemos integrar código nos templates, e ainda uma possível aplicação, mesmo que simples e rudimentar. Apesar de bastante simples, o sistema é bastante poderoso. Conseguimos utilizar as diversas classes do .NET directamente, incluir assemblagens com mais funcionalidade, e ainda definir novas estruturas a utilizar e reaproveitar nos templates.

A estrutura de processamento é algo complexo, mas permite flexibilidade no uso das transformações para aumentar as potencialidades, nomeadamente na possibilidade de definir objectos de leitura de inputs e de versões do motor de processamento. Convém entender como é feita a transformação.

Cada template que criamos é transformado numa class que é posteriormente instanciado para efectuar a produção do ficheiro de saída. Analizando o exemplo da data/hora e a classe transformada:

```
namespace
A7eae43e34a4a4ca88be15d4f32fa114c {
    using System;

    public class HelloWorld :
Microsoft.VisualStudio.TextTemplating.T
extTransformation
        {

        public override string
TransformText() {
            try {
                this.Write("\r\n");
                this.Write("\r\n");
                this.Write("\r\n<html>\
r\n    <body>\r\n       Olá Mundo! Fui
criado às ");

this.Write(Microsoft.VisualStudio.TextT
emplating.ToStringHelper.ToStringWithCu
lture(DateTime.Now.ToString("HH:mm")));

                this.Write("\r\n
<body>\r\n</html>\r\n\r\n");
            }
            catch (System.Exception e)
                    {
                System.CodeDom.Compiler
.CompilerError error = new
System.CodeDom.Compiler.CompilerError()
;
                error.ErrorText =
e.ToString();
```

```
                error.FileName =
"HelloWorld.tt";
                this.Errors.Add(error);

            }
            return
this.GenerationEnvironment.ToString();
        }
    }
}
```

Repara que foi gerado um namespace temporário e incluído as referências das directivas. Foi criado uma classe com o nome do nosso ficheiro e que herda de Microsoft.VisualStudio.TextTemplating.TextTransormation. Esta classe tem um método TransformText() ao qual é feito um overide. A nova definição do método inclui o texto do nosso template. Repare que o texto escrito fora dos blocos <# #> é inserido em métodos Write(). Texto em blocos de expressões <#= #> é inserido num Write() como valor de retorno do Microsoft.VisualStudio.TextTemplating.ToStringHelper.ToStringWithCulture(). Qualquer código em blocos <# #> é inserido no método TransformText() como código do mesmo. Podemos adicionar mais métodos a esta classe (HelloWorld) usando o controlo <#+ #>, como métodos auxiliares, ou novas classes se as definirmos em blocos <#+ #> de ficheiros externos incluídos pela directiva <#@ Include #>.

Esta classe é, depois de criada, instanciada pelo motor de processamento e o método TranformText() é chamado para criar o ficheiro resultante. O acesso a recursos é controlado por uma entidade conhecida por 'Host' que o motor de processamento utiliza sempre que necessita de aceder ao ambiente aplicacional. À partida temos apenas um host incluído na API de extensibilidade do VS.

Com o host original, o template já é bastante funcional e poderoso. Mas é extraordinário quando construímos os nossos próprios hosts e processadores. A possibilidade de o motor de processamento interagir com os nossos Hosts permite-nos definir estruturas de metadados próprias do domínio em que estamos a trabalhar, com validação e ferramentas mais eficientes na produção das mesmas. A mesma estrutura de classes rudimentar que definimos no exemplo pode ser construído e estendido numa DSL (Domain Specific Language) e gerado uma ferramenta de trabalho para criar o ficheiro de definições para a mesma. O host gerado alimentará o motor com a informação necessária para construirmos código ou outro tipo de recurso necessário.

Passos em frente

O uso de templates não é limitado à geração de código com o definido anteriormente. Pode ser usado para gerar documentação de um projecto, por exemplo, e que pode seguir em diversos formatos, usando templates diferentes. Pode também ser usado para processar ficheiros de texto, extraído determinados campos ou blocos para outro. Já os utilizei, por exemplo, para criar enumerações e dicionários em C# a partir de uma lista oficial de distritos e concelhos e moradas.

Também não estamos limitados a um ficheiro de saída por template. Apesar de não suportado directamente, há exemplos de métodos a incluir que permite guardar o conjunto de texto gerado até determinado momento num ficheiro. Com um template, é possível percorrer todos os objectos do domínio e criar uma classe BO, DAL, BLL, e documentação para cada um, com cada bloco gerado num ficheiro independente e integrado no projecto. Esta solução e muitas outras estão descritas por Oleg Sych no blog dele, e incluído no T4 Toolkit que ele desenvolveu. O blog deste autor é actualmente, e na minha opinião, a melhor fonte de informação acerca dos templates T4 e o T4 Toolkit e os exemplos apresentados são valiosos para maximizar o aproveitamento.

Uma outra forma de maximizar a potencialidade é construir uma DSL (Domain Specific Language). A Microsoft disponibiliza os Visual Studio DSL Tools para construir DSLs gráficos, integrados no VS, melhorando a experiência de produzir os metadados. O Class Designer já existente no VS é um bom exemplo de uma DSL integrada. Um bom livro a seguir é o "Domain Specific Development with Visual Studio DSL Tools". Não é um tema leve, mas as potencialidades justificam o esforço. O blog do Hugo Ribeiro também é uma boa fonte de dados.

Por fim, é importante referir que o Visual Studio tem um "defeito" relacionado com os templates T4. Não inclui suporte de cor e intelisense à sintaxe dos templates. A falta de coloração e instelisense torna a produção dos templates um processo duro. Felizmente há um add-in chamado T4-editor, criado pela Clarius Consulting, que existe na versão Profissional e comunitária (gratuita mas com algumas limitações). A versão profissional é bastante em conta (cerca de 90 euros) e justifica o custo facilmente. No mínimo, justifica integrar a versão comunitária para melhorar a experiência.

Bibliografia e ligações externas

• Visual t4 editor
http://www.visualt4.com/downloads.html

• Blog com uma das fontes mais ricas acerca de T4
http://www.olegsych.com/

• Gareth J's weblog
http://blogs.msdn.com/garethj/default.aspx

•Tim cools blog
http://www.timcools.net/post/2009/03/09/Template-based-code-generation-with-T4.aspx

• Microsoft Domain-Specific Language (DSL) Tools
http://www.microsoft.com/downloads/details.aspx?familyid=57A14CC6-C084-48DD-B401-1845013BF834&displaylang=en

•Blog do Hugo Ribeiro
http://arquitecturadesoftware.org/blogs/hugoribeiro

SOBRE O AUTOR

Miguel Alho é licenciado em Engenharia de Electrónica e Telecomunicações pela Universidade de Aveiro. Actualmente, reside e trabalha na Murtosa, como freelancer, e desenvolve aplicações e serviços web para entidades nacionais e internacionais, utilizando principalmente a tecnologia .NET da Microsoft.

miguel.alho@portugal-a-programar.org

Miguel Alho

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar @portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org