PROGRAMAR

REVISTA PORTUGUESA DE PROGRAMAÇÃO • WWW.PORTUGAL-A-PROGRAMAR.ORG

EDIÇÃO #25 -SETEMBRO 2010

ISSN 1647-0710

JOGO DO PONG COM SLICK2D (JAVA)

LUA 5ª PARTE : LINGUAGEM DE PROGRAMAÇÃO

FLEX e BYACC

SOFTWARE OPEN SOURCE EM SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

índice

equipa PROGRAMAR

coordenadores
Fernando Martins

editor
António Silva

capa
Sérgio Alves

redacção
André Silva
Augusto Manzano
João Matos
João Mares
Jorge Paulino
Marco Afonso

equipa de revisão
Fernando Martins
José Oliveira
Liliana Baptista
Sérgio Lopes

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Dados dados...

Com a denominada revolução da Web 2.0, a utilização que muitos davam à internet foi completamente modificada. O utilizador comum passou de um simples leitor e recolector a um dos mais importantes factores na Web. Hoje é possível encontrar blogs sobre quase todos os assuntos, que reflectem muitas vezes uma simples opinião pessoal e não uma verdade absoluta.

Todavia, mais perigoso que a existência de informação errada é a existência de excesso de informação sobre cada um. Isto levanta a questão de para que verdadeiros fins poderão ser utilizados os dados lá colocados. De certeza que se lembra das famosas "Perguntas Secretas", o sistema utilizado pelos serviços de correio electrónico para podermos redefinir a nossa senha em caso de esquecimento. Perguntas como "Qual foi o meu primeiro cão?" ou "Qual é a minha cor preferida?". Actualmente um dos meios mais práticos para descobrir esses mesmos dados, é fazer uma pesquisa pelo Hi5, Facebook e quem sabe não será fácil descobrir uma história "apaixonada" sobre a vida do seu primeiro cão? Ou então descobrir que todas as personalizações feitas às páginas são de um determinado tom de cor? Isso são dados pessoais dados, apesar de muitas vezes não termos noção da real utilidade que os mesmos podem ter.

Recentemente a CNPD (Comissão Nacional de Protecção de Dados) impediu a Google Portugal de recolher mais imagens para o seu serviço Street View de outras cidades portuguesas ainda não contempladas, até ser informada sobre a ferramenta utilizada para garantir a privacidade das pessoas (desfocando caras e matrículas), e que esforços está a fazer a Google para a melhorar de modo a não ser tão falível como actualmente.

«É impossível apagar o seu passado online e seguir em frente.»

Também recentemente o director executivo da Google, Eric Schmidt, afirmou: "Penso que a sociedade não entende o que acontece quando tudo está disponível, publicado e gravado por todos, o tempo inteiro." Esta questão levantou muitas críticas, principalmente por parecer defender ideias contrárias à da própria Google. No contexto destas declarações um colunista do New York Times escreveu: "É impossível apagar o seu passado online e seguir em frente." No entanto Eric Schmidt ainda foi mais longe ao afirmar: "Mostre-nos 14 fotografias suas e nós vamos identificá-lo. Acha que não há 14 fotografias suas na internet?"

Também após muita polémica, e trocas de acusações, no inicio deste ano lectivo 700 escolas vão passar a ter videovigilância instalada, mas para garantir a privacidade, a CNPD estabeleceu regras para a instalação das câmaras nas escolas. Não podem estar direccionadas para zonas de recreio e salas de aula, mas apenas em locais de acesso à escola e nas suas imediações.
Isto mostra sem dúvida a preocupação crescente sobre aquilo que podemos partilhar sem que amanhã nos arrependamos disso.

Nota: A Revista PROGRAMAR, depois de um acordo com a FCA, irá disponibilizar análises a algumas das publicações da FCA, nomeadamente livros directamente relacionados com a programação. O nosso intuito é criar análises rigorosas e isentas que demonstrem os pontos positivos e negativos das publicações. Estamos também a preparar outras parcerias com o intuito de lhe trazer a si, o nosso leitor, uma melhor e maior informação.

António Silva

Mozilla lança beta 4 do Firefox 4

http://www.mozilla.com/pt-PT/firefox/all-beta.html

Actualmente na beta 4, a Mozilla pretende lançar a versão final do Firefox 4 até ao final de Novembro deste ano.

A versão 4 irá trazer algumas novidades, sendo que as que mais saltam é vista é a nova disposição por defeito dos separadores (em que os separadores aperecem por cima da barra de endereços e dos botões) e a existência de um botão "Firefox", ao invés do habitual menu. No entanto nesta nova versão beta do Firefox também são incluídas novas funcionalidades, como um novo modo de organização de tabs, e um modo de sincronização de marcadores, histórico, separadores abertos, definições...

Google Chrome 6 Lançado

http://www.google.com/chrome

Segundo a Google a nova versão do browser é duas vezes mais rápida que a anterior e foram corrigidas 17 falhas de segurança, 8 das quais eram consideradas de risco alto.

Windos Phone 7 já está terminado

A Microsoft já terminou o Windows Phone 7, e agora só falta esperar que as operadoras terminem os pormenores finais para vermos smarthpones com o novo OS.

No Windows Phone Blog pode ler-se que a equipa do Windows Phone está "entusiasmada por ter atingido o maior marco da nossa equipa interna: a disponibilização para fabrico (RTM) do Windows Phone 7".

O que isto significa é que o sistema operativo em si está concluído, apesar de o trabalho de integração com os parceiros da Microsoft, tanto no campo do hardware, como do software e operadoras, continuar.

A Microsoft diz que o Windows PHone 7 é a sua plataforma móvel mais testada de sempre, com quase dez mil dispositivos a correr testes automatizados numa base diária, e mais de meio milhão de horas de utilização.

Apesar de a empresa de Redmond ainda não ter avançado com uma data para a disponibilização no mercado dos smartphones com este sistema operativo, o consenso na Internet é que tal deverá acontecer por volta de outubro.

IOI 2010 - International Olympiad in Informatics

http://www.ioi2010.org/index.shtml

http://www.dcc.fc.up.pt/oni/2010/

Infelizmente nenhum dos repersentantes portugueses nas IOI conseguiu a tão ambicionada medalha, apesar dos esforços. Resta-nos, assim, agradecer a toda a delegação portuguesa pelo seu esforço em levar o nome de Portugal mais longe, e ainda com mais atenção para os Professores Pedro Ribeiro e Pedro Guerreiro, pelo seu empenho em garantir todos os anos uma delegação portuguesas nas edições da IOI, apesar de todos os obstáculos com que se deparam.

Não te esqueças, esta página pode ser tua!
http://www.revista-programar.info/front/yourpage

Pong com Slick2D em Java

Neste artigo, vamos aprender a utilizar uma simples biblioteca para jogos 2D chamada Slick2D. Esta biblioteca é bastante simples de usar e abstrai o programador das partes mais aborrecidas na implementação básica de um jogo, tais como a implementação do ciclo principal de jogo e comunicação com o sistema operativo.

Vou explicar o código da minha implementação, assim como os conceitos fundamentais no desenvolvimento de qualquer jogo. Para começar vamos estudar a estrutura fundamental de qualquer aplicação interactiva.

Um jogo, num nível mais abstracto, é uma aplicação que recebe e responde a eventos. Esses eventos podem ser externos (teclas pressionadas, movimento do rato) ou internos (por exemplo num jogo de carros, detectar a colisão contra uma parede). Em termos de código, isto traduz-se em algo semelhante a:

```
bool gameIsRunning = true;

while( gameIsRunning ) {
    update();
    render();

    if( userWantsToQuit() )
        gameIsRunning = false;
}
```

Enquanto a varíavel que indica se o jogo está a decorrer (gameIsRunning) o ciclo while é sempre executado. Em cada ciclo, actualizamos o estado do jogo e desenhamos o mundo para ecrã. Se o utilizador entretanto fechar a janela ou carregar numa tecla pré-definida para o efeito, a função userWantsToQuit retorna verdadeiro, actualizamos a variável de saida, e o ciclo acaba.

É fundamental que percebam este conceito, pois é a base de qualquer jogo. Numa aplicação mais complexa, existem mais pormenores a ter em atenção, mas para este jogo, a biblioteca abstrai grande parte do essencial.

Para começar, vamos criar um novo ficheiro em Java, importar a biblioteca Slick. Podem obter a biblioteca no site oficial localizado em http://slick.cokeandcode.com/. Eu vou utilizar a biblioteca no ambiente NetBeans IDE, mas não devem ter problemas com qualquer outro ambiente de desenvolvimento para Java.

Para dar uso da biblioteca Slick2D vamos criar uma classe para representar o jogo. Eu, num momento de grande originalidade, chamei a classe 'Game'. Temos também de definir alguns métodos na nossa classe, que serão chamados pela biblioteca nos momentos apropriados.

```
package mongo;

import org.newdawn.slick.*;

public class Game extends BasicGame {

    public Game(String title) {
        // Para implementar
    }

    public void init(GameContainer gc)
{
        // Para implementar
    }

    public void update(GameContainer
gc, int delta) {
        // Para implementar
    }

    public void render(GameContainer
gc, Graphics g) {
        // Para implementar
    }

    public static void main(String[]
args) {
       // Para implementar
    }
}
```

A classe principal vai extender a classe BasicGame, disponibilizada pela biblioteca. Esta é a estrutura base para qualquer jogo desenvolvido com a biblioteca Slick2D. Agora temos de escrever o código para iniciar a biblioteca e começar o ciclo principal de jogo.

```
public static void main(String[] args)
{
    AppGameContainer app = new
AppGameContainer(new Game(title));
    app.setDisplayMode(width, height,
fullscreen);
    app.setTargetFrameRate(fpslimit);
    app.start();
}
```

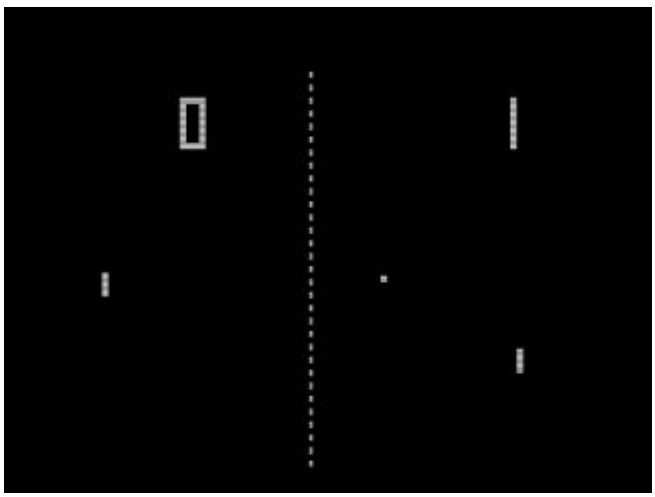
Este código cria uma nova instância da aplicação e passa a classe correspondente ao jogo no constructor. O título pretendido (que queremos que apareça na janela) é passado como argumento do constructor da classe do jogo. Depois iniciamos a janela com a largura e algura desejada, e escolhemos se a janela vai ocupar o ecrã todo. Para finalizar, executamos o método que vai iniciar o ciclo principal do jogo. Como podem ver no código, os valores foram definidos como variáveis estáticas da classe, para facilitar futuras alterações.

```
static int width = 640;
static int height = 480;
static boolean fullscreen = false;

static String title = "Mongo";
static int fpslimit = 60;
```

Entretanto já temos definida a estrutura básica de uma aplicação, e se executarem a aplicação deve ser mostrada uma janela sem conteúdo. Para mostrar o conteúdo temos de implementar os métodos correspondentes ao ciclo de actualização e desenho da aplicação.

Vamos então pensar como modelar o jogo do Pong. Este jogo foi um dos primeiros video jogos e consiste numa simulação de um campo (de ténis de mesa), onde uma bola é atirada de um lado para o outro. O jogo é normalmente jogado por 2 jogadores, mas a nossa versão vai também ter uma opção onde o segundo oponente é controlado pelo computador.

Para representar as duas raquetes, podemos utilizar imagens, mas para facilitar a implementação vamos utilizar dois rectângulos. Para a bola, vamos utilizar um círculo. A biblioteca Slick2D já disponibiliza estas primitivas geométricas.



Pong (1972) - Atari Inc.

```
Circle ball;

Rectangle paddlePlayer;
Rectangle paddleCPU;
```

Para mover a bola pelo ecrã, temos de representar a sua velocidade. Estas quantidades são normalmente representadas por vectores de duas dimensões. A biblioteca Slick2D possui também uma biblioteca de matemática com estas primitivas implementadas. Vamos definir algumas variáveis para a velocidade e para armazenar as pontuações obtidas por cada jogador.

```
Vector2f ballVelocity;

int scorePlayer;
int scoreCPU;
```

Já temos, então, tudo o que precisamos para guardar o estado de jogo. Agora falta implementar os vários métodos que ficaram em branco. Vamos começar com o método de iniciação, que é chamado pela biblioteca quando o jogo é iniciado. Este método tem um parâmetro do tipo GameContainer, através do qual podemos aceder a outras classes que oferecem vários serviços, como o sistema de input, contexto dos gráficos 2D, sistema de som, etc.

```
public void init(GameContainer gc) {
    gc.getInput().enableKeyRepeat();

    paddlePlayer = new
RoundedRectangle(5, height/2, 10, 80,
3);
    paddleCPU = new
RoundedRectangle(width-15, height/2,
10, 80, 3);
    ball = new Circle(width/2,
height/2, 6);
    ballVelocity = new Vector2f(-3, 1);
}
```

Neste métodos vamos ligar suporte para repetição de teclas (assim enquanto uma tecla estiver pressionada, o Slick2D envia sempre eventos de input). Depois iniciamos as variáveis de jogo declaradas anteriormente. Os constructores das figuras recebem os tamanhos e posições na janela. As duas raquetas são criadas na esquerda e na direita, e depois a bola é criada no centro do ecrã, com velocidade horizontal inicial correspondente a

deslocação para a esquerda e para baixo.

Depois da iniciação estar concluida, falta implementar a actualização e a renderização do jogo.

```
public void render(GameContainer gc,
Graphics g) {
    g.fill(paddlePlayer);
    g.fill(paddleCPU);
    g.fill(ball);
}
```

O código cima é muito simples e só desenha as figuras que iniciámos para representar os jogadores e a bola. Com o código escrito até agora, o jogo já pode ser iniciado e são desenhadas todas as figuras do jogo. Falta então adicionar vida ao jogo, com a função de simulação/actualização (o método update).

O parâmetro delta é usado para que a simulação do jogo seja independente da velocidade de renderização. Vejam o link nos recursos que explica diferentes implementações de game loops. Como este jogo é muito simples, vamos ignorar o delta.

Primeiro vamos pensar no que pode acontecer no Pong. Se o utilizador carregar nas teclas temos de mover o rectângulo para cima ou para baixo, dependendo da tecla que foi pressionada.

```
if
(gc.getInput().isKeyDown(Input.KEY_UP))
{
    if(paddlePlayer.getMinY() > 0)

paddlePlayer.setY(paddlePlayer.getY() -
10.0f);
} else if
(gc.getInput().isKeyDown(Input.KEY_DOWN)
) {
    if(paddlePlayer.getMaxY() < height)

paddlePlayer.setY(paddlePlayer.getY() +
10.0f);
}
```

Temos também de actualizar a posição da bola com a respectiva velocidade.

```
ball.setLocation(ball.getX()+ballVelocit
y.getX(),
    ball.getY()+ballVelocity.getY());
```

Agora vamos detectar as colisões que podem acontecer quando a bola bate nos limites do campo de jogo. Em termos horizontais, se a bola sai do ecrã pela esquerda ou pela direita, temos de trocar a velocidade horizontal e actualizar as pontuações.

```
if(ball.getMinX() <= 0) {
    ballVelocity.x = -
ballVelocity.getX();
    scoreCPU++;
}

if(ball.getMaxX() >= width) {
    ballVelocity.x = -
ballVelocity.getX();
    scorePlayer++;
}
```

Agora em termos verticais, só temos de trocar a velocidade na componente Y do vector da velocidade.

```
if(ball.getMinY() <= 0)
    ballVelocity.y = -
ballVelocity.getY();

if(ball.getMaxY() >= height)
    ballVelocity.y = -
ballVelocity.getY();
```

Resta detectar colisões quando a bola bate nas raquetes. Basta trocar a velocidade na componente do X.

```
if(ball.intersects(paddlePlayer) ||
ball.intersects(paddleCPU)) {
    ballVelocity.x = -
ballVelocity.getX();
}
```

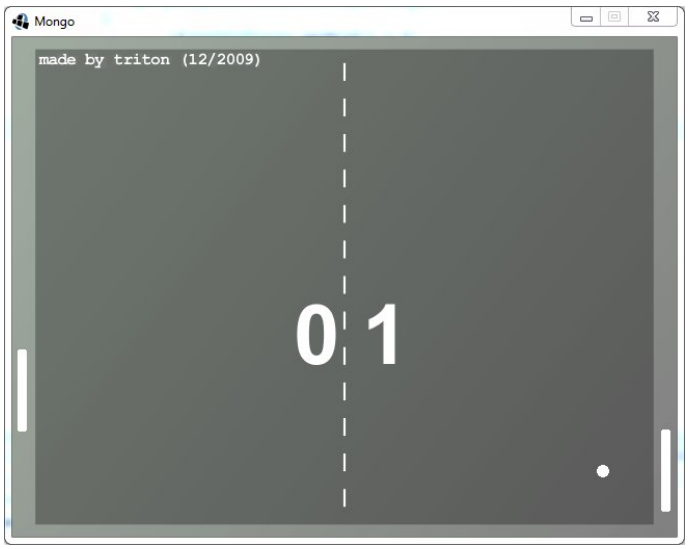
Experimentem correr o jogo e já devem ter algo que se assemelha ao jogo do Pong. Mas temos uma falha grande. A raquete do oponente não é simulada pelo computador. Vamos adicionar ao jogo, para podermos jogar contra o computador. Na minha implementação a "inteligência artificial" é muito simples. A raquete do oponente segue sempre a posição vertical da bola, tornando impossível vencer o computador:

```
float posY = ball.getCenterY()-
paddleCPU.getHeight()/2;
paddleCPU.setY(posY);
```

Muito simples! O algoritmo pode ser melhorado através do uso de números aleatórios. Experimentem modificar o jogo e adicionar novas funcionalidades.

Recursos

- http://dev.koonsolo.com/7/dewitters-gameloop/
- http://slick.cokeandcode.com/
- http://netbeans.org/
- http://slick.cokeandcode.com/wiki/doku.php



SOBRE O AUTOR

João Matos, também conhecido por triton, frequenta o segundo ano do curso de Engenharia Informática e Computadores no IST. Tem como principais interesses as áreas de Sistemas Operativos, Computação Gráfica e Arquitectura de Computadores.

João Matos

Software Livre em Sistemas de Informação Geográfica

Introdução

Um Sistema de Informação Geográfica é a aplicação da área interdisciplinar Ciência da Informação Geográfica, de modo em que cruza, trata e analisa informação produzida por várias áreas tais como: Geografia, Cartografia, Detecção Remota (informação obtida por satélite), Topografia, Fotogrametria, ordenamento do território, gestão de catástrofes naturais e sistemas de navegação[1].

Sistemas de Coordenadas e Projecções[2]

Antes de avançar mais concretamente para o tema central deste artigo, temos de falar um pouco sobre um conceito essencial em SIG: coordenada. Não é do âmbito deste artigo mergulhar nesta matéria - até porque entramos no domínio da matemática / física e não é isso que se pretende - mas é importante deixar o leitor consciente de que existem vários sistemas de coordenadas, bem com várias projecções e a conversão entre sistemas de coordenadas e projecções é uma constante no dia-a-dia do profissional que trabalha nestas matérias. Como é lógico, hoje em dia, nenhum destes cálculos matemáticos se fazem manualmente e por isso existem várias ferramentas que realizam estes cálculos.

O sistema de coordenadas mais conhecido é o UTM - Universal Transverse Mercator - que serve para indicar posições absolutas no globo terrestre. Neste sistema existem 3 coordenadas: Latitude, Longitude e Altitude - o último é desprezível em contexto de duas dimensões.

Um dos serviços online que podemos usar para efectuar conversão de coordenadas entre diferentes projecções pode ser encontrado no web site http://www.faunalia.pt/conversao.

Contudo, ao trabalhar com dados em massa, por exemplo um mapa com milhares de pontos, não é viável converter coordenadas em vários ficheiros desta forma e por isso, uma das bibliotecas - talvez a mais importante - que utiliza métodos de conversão e que facilmente se integra em qualquer aplicação FOSS, é a PROJ.4[3]

Formatos de dados

Pouco a pouco vamos introduzindo conceitos básicos usados em SIG, como é o caso dos 2 tipos de dados básicos: Vector e Raster.

- Vector - Normalmente estruturado numa lista de vértices, e como consequência o consumo de memoria virtual ao processar, bem como o espaço necessário ao armazenar, é reduzido.
- Raster - Refere-se a um varrimento de uma área que depois é estruturada numa matriz. Ao contrário do vector, esta estrutura aumenta consideravelmente o consumo de memória virtual ao processar, bem como o espaço necessário ao armazenar.
- Representação de Vectores e Raster - Tanto vectores como rasters são necessariamente armazenados em ficheiros ou bases de dados. Segue-se uma breve descrição sobre os formatos de ficheiros mais usados:
  - KML / KMZ - formato usado para apresentar dados geográficos;
  - GPX - formato de dados XML para representar dados de GPS;
  - GML - formato de dados XML para exprimir características geográficas.
  - WKT - é uma linguagem de texto para representar geometrias, sistemas de referência espacial e transformações entre sistemas de referência espacial. Usado essencialmente em bases de dados.
  - GeoJSON - formato para codificar estruturas de dados geográficas.
  - Geotiff - formato baseado no Tiff para intercâmbio de dados raster contendo dados geográficos.
  - JPG, PNG e GIF - tipos de ficheiros de imagem sem dados geográficos.
  - SVG - também baseado no XML, serve para descrever de forma vectorial desenhos e gráficos bidimensionais.
  - EPS - desenvolvido pela Adobe, formato digital para imagens.

Caminho mais rápido mas limitado

A boa notícia: existe um caminho rápido para atingir como objectivo a disponibilização de informação geográfica através de um mapa. Como todos nós já vimos, existem na Internet ferramentas ou plataformas completas que suportam operações comuns em Sistemas de Informação Geográfica, nomeadamente o Google Maps, da Google. Esta plataforma vêm acompanhada de uma API que permite a qualquer utilizador, mesmo sem experiência em desenvolvimento de aplicações SIG, criar o seu próprio mapa, personalizá-lo e partilhá-lo no seu website, bastando para isso copiar alguns exemplos documentados na própria API da Google Maps.

Para os leitores mais impacientes, deixamos abaixo algumas linhas de código que facilmente se integram numa página HTML e que permitem a visualização de um mapa acompanhado de ferramentas de navegação.

O exemplo mais simples intitulado "Hello World"[4]:

```
<!DOCTYPE html "-//W3C//DTD XHTML 1.0
Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1
-strict.dtd">
<html
xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="content-type"
content="text/html; charset=utf-8"/>
   <title>Google Maps JavaScript API
Example - Tavira</title>
   <script
src="http://maps.google.com/maps?file=a
pi&amp;v=2&amp;key=abcdefg&sensor=true_
or_false"
        type="text/javascript"></script
>
   <script type="text/javascript">

   function initialize() {
    if (GBrowserIsCompatible()) {
    var map = new
GMap2(document.getElementById("map_canv
as"));
    map.setCenter(new
GLatLng(37.148605,-7.64946), 12);
    map.setUIToDefault();
    }
   }

   </script>
 </head>
 <body onload="initialize()"
onunload="GUnload()">
   <div id="map_canvas" style="width:
990px; height: 600px"></div>
 </body>
</html>
```

A má notícia: desta forma ficamos confinados e limitados às ferramentas desta plataforma. A título de exemplo, no Google Maps é possivel usar 3 tipos de mapas:

1. Terreno - onde se distinguem estradas, lagos, fronteiras e reservas naturais;
2. Satélite - imagens fornecidas por satélite; é também possível activar a visualização de etiquetas.
3. Street View - vista de rua; uma visualização que proporciona um ambiente real em determinadas localizações.

Contudo, para uma organização que detêm muita informação própria em suporte de base de dados, centenas de ficheiros raster e shape, a utilização da API do Google Maps torna-se ineficiente para lidar com tamanha informação - para não falar da necessidade de edição de raster e vectores. Assim, surge a necessidade de usar o próprio servidor de mapas, WMS (Web Map Service) e um conjunto de ferramentas de software Livre que vamos ver a seguir.

O caminho mais lento contudo... cheio de potencialidades!

Primeiro a "má" notícia - se é que se pode admitir como uma má notícia, uma vez que esta se verifica em qualquer ambiente em produção de soluções empresariais: é necessário um forte investimento em formação contínua dado que no âmbito de desenvolvimento de software é muita a frequência com que surgem novas soluções.

A boa notícia é que existe uma panóplia de software Livre - apartir de agora será referido como FOSS (Free and Open Source Software) neste artigo - "para todos os gostos", permitindo assim qualquer profissional individual, ou pequena empresa, atingir objectivos que apenas estariam ao alcance de grandes empresas.

Passando finalmente ao cerne da questão, apresenta-se uma breve descrição de FOSS mais usado em SIG, enquadrado em 3 grande categorias: Sistemas de Gestão de Bases de Dados, Desktop e Web.

Nos Sistemas de Gestão de Bases de Dados (relacionais) em FOSS, destaca-se Postgres (incluindo a extensão geoespacial PostGIS) e MySql. Não é possível debruçar-nos neste artigo sobre um estudo comparativo entre estes dois sistemas mas pode-se indicar algumas leituras sobre essa matéria[5].

Em FOSS para Desktop usado em SIG destaca-se: QuantumGIS[6], Kosmo[7], gvSIG[8], OpenJUMP[9] e GRASS[10].
É extremamente difícil apresentar neste artigo um estudo comparativo sobre os programas indicados acima, pelo que deixamos os respectivos links para que o leitor daí retire as suas próprias conclusões.

De qualquer forma não se pode deixar de referir que embora alguns deles sejam redundantes em certas funcionalidades, a grande valia está na forma como se complementam; o que um não faz, o outro faz. Isto faz com que, inevitavelmente num desenvolvimento extremo todos eles sejam importantes.

Em FOSS para aplicações web, ou por vezes denominado

por websig ou webgis, encontramos também poderosíssimas plataformas de suporte e apresentação de dados geoespaciais.
Mapserver[11] - Servidor de mapas baseado em PHP;
Geoserver[12] - Servidor de mapas baseado em JAVA;
OpenLayers[13] - Plataforma em Javascript;
Mapbender[14] - Plataforma em PHP, Javascript e XML;
Mapfish[15] - Plataforma em Python, baseada no Pylons. Combina várias frameworks Javascript, nomeadamente; OpenLayers, ExtJS e GeoExt.

Como podemos notar, existe uma forte componente de Javascript nestas plataformas clientes. Isto deve-se essencialmente ao facto de que as páginas web com mapas não se actualizam totalmente quando o visitante interage com a aplicação, recorrendo quase sempre a AJAX[16].

Existem ainda bibliotecas "obrigatórias" usadas por vários ambientes SIG e que sem elas não era simplesmente possível produzir aplicações e resultados geográficos em larga escala. As mais usadas são: Gdal/Ogr, Proj.4, Geotools e FWTools.

Em ambientes Windows, existe o pacote OSGEO4W[17] que contém uma colecção de software (alguns indicados acima), fácil de instalar e que permite iniciar rapidamente desenvolvimento em SIG sem que o utilizador se preocupe com dependências.

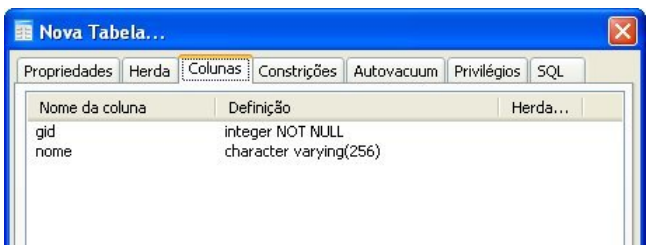
Para terminar este capítulo, deve-se ainda referir que, tal como existe a W3C[18] que especifica normas standard para todo o desenvolvimento Web, assim existe a OGC[19] para regular ou nomalizar todo o desenvolvimento FOSS em Sistemas de Informação Geográfica.

Caso de Estudo - Percurso Ciclo-turismo

Postgres

Criar base de dados - aproveita-se esta oportunidade para demonstrar a Interface gráfica do pgadmin, sendo este também um software opensource.
Para este caso foi criado o seguinte:

- Nova base de dados "sig_revista"
- Novo esquema "casestudy1" na base de dados indicada anteriormente.
- Nova tabela "percurso" com os seguintes campos:

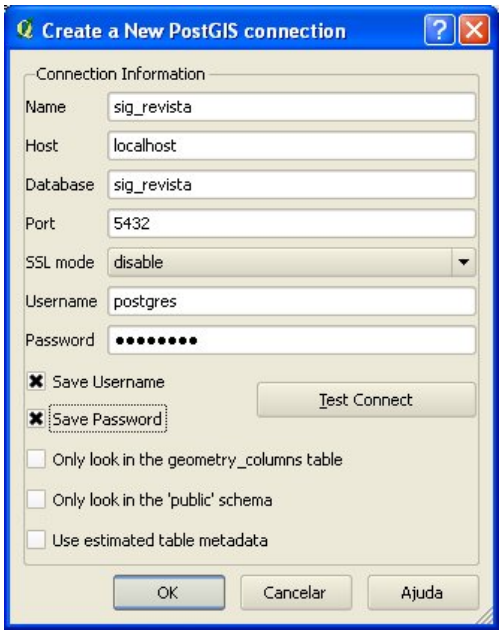


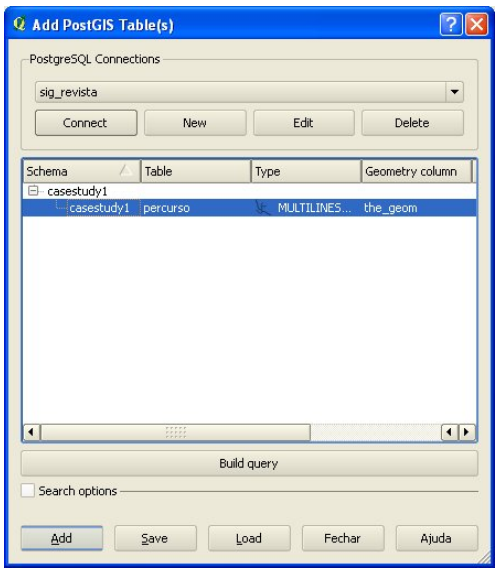
- Para adicionar uma coluna de geometria no postgres e para que os vários clientes desktop possam interagir com o postgres, é necessário usar a função AddGeometryColumn. Numa janela SQL, executar:

```
SELECT AddGeometryColumn('casestudy1', 'percurso', 'the_geom', 27492, 'MULTILINESTRING', 2 );
```

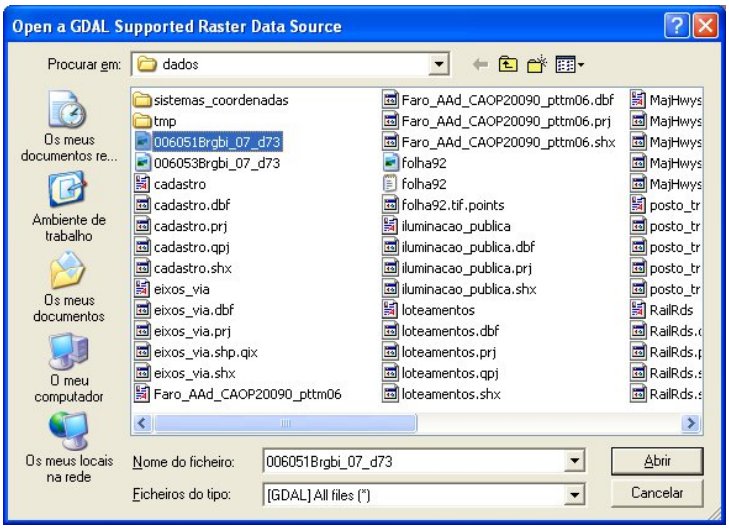
QuantumGIS

Passando agora às aplicações Desktop, vamos construir o nosso mapa. Para começar, podemos interagir com a base de dados postgres e associar uma camada (layer) à nossa recém nascida tabela "percurso". No QuantumGIS, clica-se no botão Add Layer PostGis e obtemos a seguinte formulário:





Precisamos também adicionar uma imagem de satélite, por um lado para tornar o nosso mapa mais atractivo, por outro lado, para obter uma ideia visual sobre o posicionamento dos objectos no mapa. Clica-se em Add Layer Raster (carregar um ortofotomapa - ficheiro geotiff).
Um ortofotomapa é uma fotografia aérea obtida por Detecção Remota, neste caso por Satélite.
O leitor deve estar a perguntar porquê o formato GeoTIFF e não um JPEG, PNG ou GIF. A razão deve-se, como já foi explicado acima, ao facto deste formato conter preciosíssima informação geográfica que lhe permite ser aceite como camada raster.
A boa notícia é que se pode converter qualquer JPEG, PNG, ou GIF, em GeoTiff, bastando para isso adicionar uma estrutura com informação geográfica. Este processo pode ser feito de várias formas, nomeadamente, usando a biblioteca Gdal/Ogr.

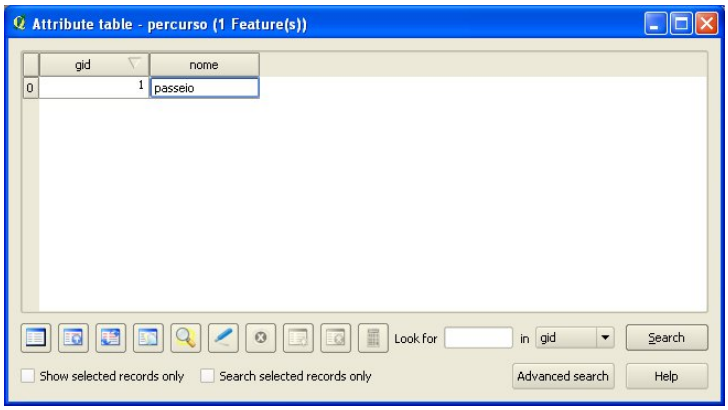


Vamos também adicionar vectores que representam os eixos de via do nosso local escolhido. Os eixos de via são importantes porque queremos desenhar o nosso percurso de Ciclo-turismo com base na rede viária do nosso local. Clica-se no botão Adicionar Layer Vector - Ficheiro com os vectores dos eixos de vias de trânsito (uma shape file por exemplo).
A shapefile é "a excepção à regra" neste artigo sobre FOSS, uma vez que é um formato desenvolvido pela ESRI e não é Livre.

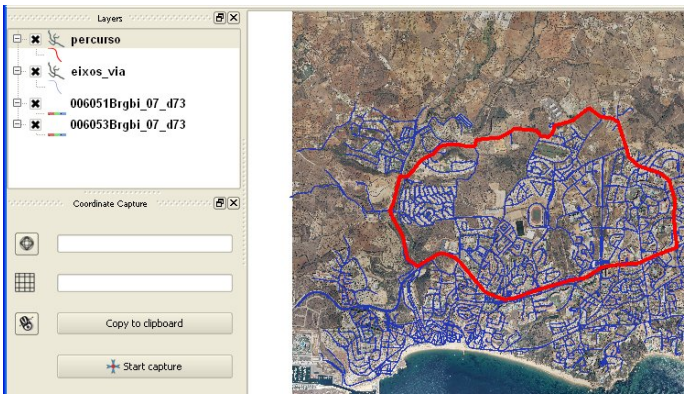
Chegando a este ponto, já temos 4 camadas no nosso mapa, sendo duas delas dois raster que não estão agregados numa só camada, ou layer.

Como não se sabe que coordenadas vão traçar o nosso percurso, faz-se um desenho no nosso mapa, na camada PostGIS. Os dados a serem gravados na nossa tabela conseguem-se com os seguintes passos:

- Iniciar o modo edição na layer "percurso";
- Desenhar o percurso usando o botão "Capture Line";
- Abrir "Attributes Table" e escrever os valores "1" e "passeio" na tabela. Terminar o modo de edição



Segue-se um screenshot do nosso mapa final trabalhado no QuantumGIS.



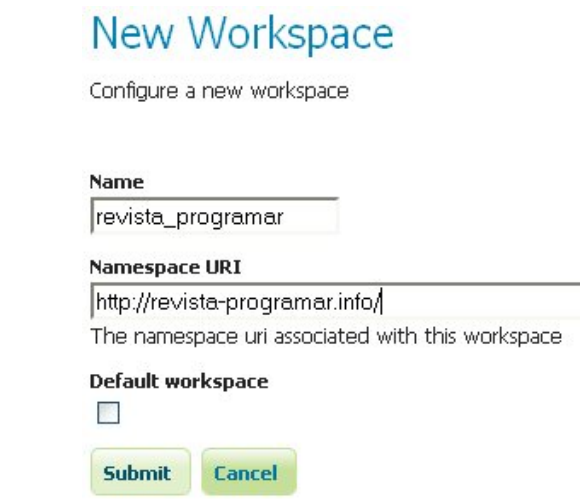
A partir daqui, já é possível por exemplo, compor uma versão de impressão com detalhes de etiquetas indicando que significa cada objecto no nosso mapa, e por conseguinte, distribui-lo à organização do evento Ciclo-turismo. Os ciclistas ficaram muito satisfeitos :)

Aplicação Web (uma entre muitas soluções possíveis)

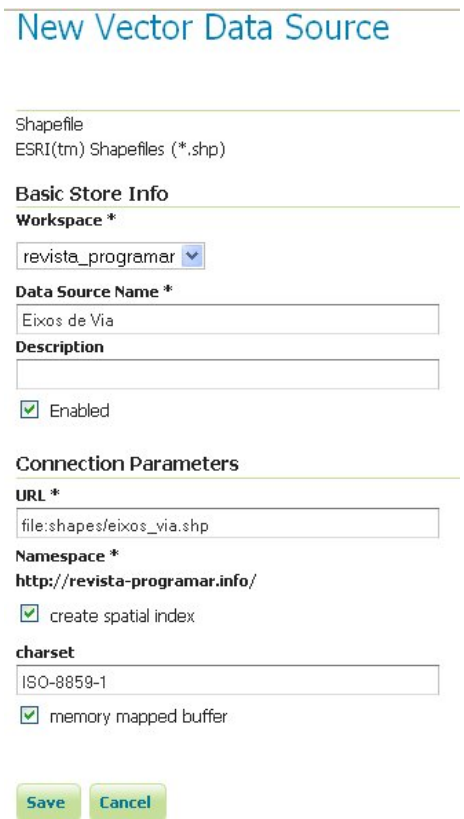
Servidor de mapas - Geoserver

O servidor de mapas escolhido, sem nenhuma razão em particular, para a nossa aplicação web é o Geoserver. Como alternativa, pode-se usar também o Mapserver.
Considerando que o Geoserver está instalado e a correr no servidor, seguem-se os seguintes passos:
Entrar na interface web através do endereço: http://localhost:8081/geoserver
Fazer login como user "admin" e password "geoserver"

O Geoserver organiza os dados em Workspaces, Stores, Layers e Styles.
O primeiro passo é configurar uma workspace.



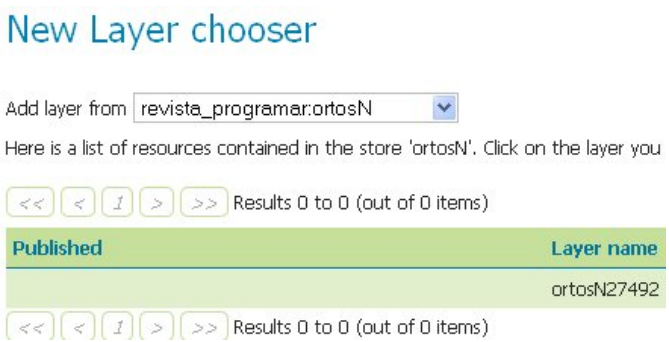
A seguir, adicionamos 4 stores, que se vão reflectir nas nossas 4 camadas como se fez no QuantumGIS.
Adicionar stores:



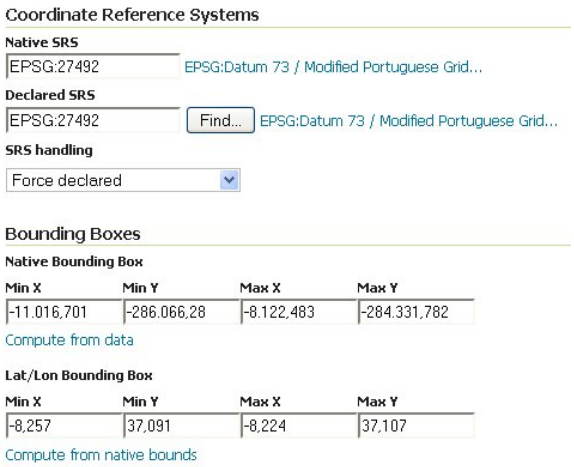
Adicionar store PostGIS. Tal como se fez para as stores anteriores mas desta feita, o Vector Data Source é PostGIS.

Layers

Após concluir a adição das stores, é tempo de configurar as camadas (layers) para que estejam activas e disponíveis para que o cliente web, por exemplo, OpenLayers, possa fazer a pedidos destas camadas ao Geoserver.



A imagem que se segue exemplifica alguns parâmetros extremamente importantes aquando a configuração de layers. Se estes parâmetros não estiverem correctamente definidos, não será possível compor correctamente o nosso mapa no lado do cliente.
Todas as stores que adicionamos partilham do mesmo sistema de Coordenadas, identificado por "EPSG:27492". Esta condição de ter o mesmo sistema de coordenadas entre as layers que queremos publicar é imprescindível para que as camadas se sobreponham correctamente.
Os dados para as Bounding Boxes, se os dados estiverem todos com informação geográfica, bastará clicar no botão "Compute from Data".



Para terminar no lado do servidor, o Geoserver irá disponibilizar as camadas (layers) através do WMS, ou Web Map Service. WMS é um web service que responde a pedidos de clientes de mapas, nomeadamente, OpenLayers. Tal como se viu no exemplo do Google Maps, agora temos o nosso WMS fornecido pelo nosso servidor de mapas Geoserver.

Cliente WEB

Para desenvolver o cliente da nossa aplicação, é necessário ter em conta os seguintes requisitos:

- Plataforma OpenLayers
- Plataforma Prototype
- Ficheiro HTML
- Ficheiro em Javascript para fazer os pedidos ao Geoserver e também aos ficheiros PHP para obter dados do PostGIS.
- Ficheiros PHP para consultar a base de dados e devolver dados no formato JSON.

É possivel com apenas um ficheiro HTML (incluido Javascript) apresentar o nosso mapa na web, no entanto, a partir desta estrutura apresentada aqui, o programador têm uma base com muito maior potencial para desenvolver esta aplicação. Outra plataforma com enorme potencial para desenvolver aplicações web em SIG é o Mapfish. Portanto as potencialidades opensource são quase inesgotaveis.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1
-transitional.dtd">
<html
xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Case Study -
Cicloturismo</title>

     <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8" />
     <link rel="stylesheet"
type="text/css"
href="css/style.css"></link>
     <script src="js/prototype.js"
type="text/javascript"></script>
        <script src="js/OpenLayers-
2.9.1/OpenLayers.js"
type="text/javascript"></script>
        <link rel="stylesheet"
type="text/css" href="js/OpenLayers-
2.9.1/theme/default/style.css"></script
>
     <script src="js/mapa.js"
type="text/javascript"></script>
    </head>
    <body onload="init()">
        <div id="map"></div>
     <div id="scale"></div>
     <div id="location"></div>
     <p><input type="button"
value="Carregar geometria da base de
dados da tabela 'paragens'"
onclick="return
updateParagens();"></input></p>
    </body>
</html>
```

Ficheiro base para apresentar a aplicação (19 linhas)

mapa.js

Servirá para ler os WMS do Geoserver, usando Openlayers, bem como fazer pedidos AJAX aos ficheiros PHP para consultar e devolver dados do PostGIS no formato JSON (171 linhas)

```
var map;
var paragens;

// pink tile avoidance
OpenLayers.IMAGE_RELOAD_ATTEMPTS = 5;
// make OL compute scale according to
WMS spec
OpenLayers.DOTS_PER_INCH = 25.4 /
0.28;

function init(){
    format = 'image/png';
    var bounds = new
OpenLayers.Bounds(
     -11050, -286200,
     -8050, -284200
    );

    var options = {
     controls: [],
     maxExtent: bounds,
     restrictedExtent: bounds,
     maxResolution:
11.467730468750002,
     projection: "EPSG:27492",
     units: 'm',
     numZoomLevels: 5
    };
    map = new OpenLayers.Map('map',
options);

    // Layer dos ortofotomapas
    var ortos = new
OpenLayers.Layer.WMS(
     "Ortofotomapa",
"http://localhost:8080/geoserver/wms",
     {
         layers: 'Cicloturismo_ortos'
     },
     {
         singleTile: false,
         isBaseLayer: true
     }
    );

    // setup tiled layer
    var eixos_via = new
OpenLayers.Layer.WMS(
     "Eixos de Via",
"http://localhost:8080/geoserver/wms",
     {
         layers:
'revista_programar:eixos_via',
         transparent: 'true'
     },
     {
         singleTile: false,
         isBaseLayer: false
     }
    );

    var percurso_wms = new
OpenLayers.Layer.WMS(
     "Percurso",
"http://localhost:8080/geoserver/wms",
     {
         layers:
'revista_programar:percurso',
         transparent: 'true'
     },
     {
         singleTile: false,
         isBaseLayer: false
     }
    );

    var paragens_wms = new
OpenLayers.Layer.WMS(
     "Paragens",
"http://localhost:8080/geoserver/wms",
     {
         layers:
'revista_programar:paragens',
         transparent: 'true'
     },
     {
         singleTile: false,
         isBaseLayer: false
     }
    );

    // create a vector layer for
drawing
    paragens = new
OpenLayers.Layer.Vector("Paragens
Vectores");

    map.addLayers([ortos, eixos_via,
percurso_wms, paragens_wms,
paragens]);

    // build up all controls
    map.addControl(new
OpenLayers.Control.PanZoomBar({
     position: new OpenLayers.Pixel(2,
```

```
15)
    }));
    map.addControl(new
OpenLayers.Control.LayerSwitcher());
    map.addControl(new
OpenLayers.Control.Navigation());
    map.addControl(new
OpenLayers.Control.Scale($('scale')));
    map.addControl(new
OpenLayers.Control.MousePosition({eleme
nt: $('location')}));

    selectControl = new
OpenLayers.Control.SelectFeature(parage
ns,
     {onSelect: onFeatureSelect,
onUnselect: onFeatureUnselect});
    map.addControl(selectControl);
    selectControl.activate();

    map.zoomToExtent(bounds);
    map.zoomTo(1);

}

function updateParagens() {
    new
Ajax.Request("http://localhost/revista/
php/ajax_db_json.php", {
     method: 'post',
        parameters: {
            esquema: 'casestudy1',
         tabela: 'paragens'
        },
        onSuccess: function(transport)
{
            var many =
json(transport.responseText, paragens,
true);
         var s = (many > 1 ? 's' :
'');
         alert('Foram carregada' + s +
' ' + many + ' geometria' + s + ' da
base de dados');
         alert('Clique num ponto de
paragem para ver mais informação');
        }
    });
}

function json(req, layer, reset) {

    var format =  new
OpenLayers.Format.JSON();
    var wktformat = new
OpenLayers.Format.WKT();
    var arreio = format.read(req);

    var feature, i, j, wktfeat,
geomarray, feat, geom;
    if (reset)
layer.destroyFeatures();
    var type = typeof arreio;

    if (arreio.length > 0) {
        for (i = 0; i<=arreio.length -
1; i++) {
            feature = arreio[i];
            if (feature.wkt) {
                wktfeat =
wktformat.read(feature.wkt);
                type = typeof wktfeat;

                if (type == 'array') {
                    geomarray = [];
                    for(j=0; j <=
wktfeat.length - 1; j++) {
                        geomarray.push(
wktfeat[j].geometry);
                    }
                    geom = new
OpenLayers.Geometry.Collection(geomarra
y);
                }
                else {
                    geom =
wktfeat.geometry;
                }

                feat = new
OpenLayers.Feature.Vector(geom,
feature, feature.style);
                layer.addFeatures([feat
], {'silent': true});
            }
        }
    }
    return arreio.length;
}

function onPopupClose(evt) {

selectControl.unselect(selectedFeature)
;
}
function onFeatureSelect(feature) {
    selectedFeature = feature;
    popup = new
OpenLayers.Popup.FramedCloud("chicken",


feature.geometry.getBounds().getCenterL
```

```
onLat(),
     null,
     "<div style='font-
size:.8em'>Detalhes da Paragem: " +
feature.attributes.nome + "</div>",
     null, true, onPopupClose);
    feature.popup = popup;
    map.addPopup(popup);
}
function onFeatureUnselect(feature) {
    map.removePopup(feature.popup);
    feature.popup.destroy();
    feature.popup = null;
}
```

geojson.php

Class Geojson em PHP que irá ler da base de dados e devolver JSON. O objectivo básico deste ficheiro está conseguido, mas pode-se melhorar as instruções de código para transformar numa Classe de DAO (85 linhas).
A função jsonencode foi desenvolvida para funcionar nas versões PHP inferiores a 5.3. Na versão PHP 5.3 é preferível usar a função nativa json_encode.

```
<?php
class Geojson {
    var $db;
    function __construct() {
     $this->db =
pg_connect("host=localhost port=5432
dbname=sig_revista user=postgres
password=password");
    }

private function jsonhelper($value) {
        if (gettype($value) ==
'boolean') {
            if ($value) return 'true';
else return 'false';
        }
        if (gettype($value) ==
'integer') return (int) $value;
        if (gettype($value) ==
'double') return (float) $value;
        if ($value == '') return
'null';
        return '"' . str_replace('/',
'\/', $value) . '"';
}

private function jsonencode($arreio) {
        $result = '';
        $i = 0;
        if (count($arreio)) {
            $keys =
array_keys($arreio);
            //echo gettype($keys[0]);
            if (gettype($keys[0]) ==
'integer') {
                $result .= '[';

                foreach ($arreio as
$value) {
                    if
(gettype($value) == 'array')
                        $result .=
$this->jsonencode($value);
                    else $result .=
$this->jsonhelper($value);
                    $i++;
                    if ($i <
count($arreio)) $result .= ',';
    }

                $result .= ']';
            }
            else {
                $result .= '{';

                foreach ($arreio as
$key => $value) {
                    if
(gettype($value) == 'array')
                        $result .=
'"'.$key.'":'.$this-
>jsonencode($value);
                    else $result .=
'"'.$key.'":'.$this-
>jsonhelper($value);
                    $i++;
                    if ($i <
count($arreio)) $result .= ',';
                }

                $result .= '}';
            }
        }
        else $result = '[]';

        return $result;
}

function read($esquema, $tabela) {

        $sql = sprintf("
            SELECT
                gid,
             nome,
                ST_AsText(the_geom) as
wkt
```

```
                FROM %s.%s
        ", $esquema, $tabela);

        $result = pg_query($this->db,
$sql);

        $array = array();
        $i = 0;
        while ($registo =
pg_fetch_assoc($result)) {
         $array[$i] = $registo;
         $i++;
     }

        //return json_encode($array);
        return $this-
>jsonencode($array);
}

function write() {
     // Por implementar
    }
}
```

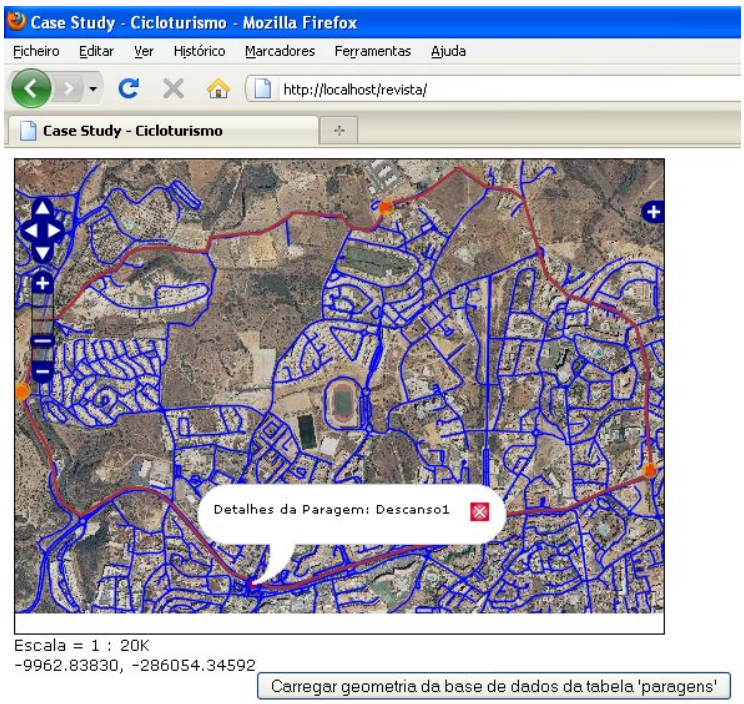
ajax_db_json.php

Servirá para receber o pedido HttpXmlRequest, usar a classe Geojson e devolver dados no formato JSON (10 linhas). Provavelmente pode-se melhorar a lógica entre este e o ficheiro geojson.php, de forma a que no ficheiro geojson.php apenas se implemente as funções da classe DAO e neste ficheiro se converta os dados obtidos para JSON.

```
<?php

header('Cache-Control: no-cache, must-
revalidate');
header('Expires: Mon, 26 Jul 1997
05:00:00 GMT');
header('Content-type:
application/json');

require_once('geojson.php');

$conn = new Geojson();
echo $conn->read($_POST['esquema'],
$_POST['tabela']);

?>
```

Aplicação Finalizada

Segue-se um screenshot da nossa aplicação web finalizada.



Avançado

Deixamos algumas dicas ao leitor, em forma de lista, para melhorar o case study demonstrado acima:

- Criar um script php para construir um ficheiro no formato GPX (baseado em XML)
- Projectar sobre o Google: colocar as layers vectoriais (percurso e paragens) sobre o WMS do Google
- É necessário definir no OpenLayers a Internal Projection para 900913.
- Carregar da base de dados as features já transformadas para 900913
- Se na tabela spatial_ref_sys não estiver definida a entrada 900913 é necessário inserir para que o PostGIS possa fazer a conversão de coordenadas correctamente:

```
INSERT into spatial_ref_sys (srid,
auth_name, auth_srid, proj4text, srtext)
values ( 900913, 'spatialreference.org',
6, '+proj=merc +a=6378137 +b=6378137
+lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0
+k=1.0 +units=m +nadgrids=@null +wktext
+ n o _ d e f s ' ,
'PROJCS["unnamed",GEOGCS["unnamed
ellipse",DATUM["unknown",SPHEROID["unnamed
",6378137,0]],PRIMEM["Greenwich",0],UNIT["
degree",0.0174532925199433]],PROJECTION["M
ercator_2SP"],PARAMETER["standard_parallel
_1",0],PARAMETER["central_meridian",0],PAR
AMETER["false_easting",0],PARAMETER["false
_northing",0],UNIT["Meter",1],EXTENSION["P
ROJ4","+proj=merc +a=6378137 +b=6378137
+lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0
```

```
+k=1.0 +units=m +nadgrids=@null +wktext
+no_defs"]]');
```

• Definir Spherical Mercator na layer WMS do Google

• Vista 3D: Grass + nviz e criar vista 3D

• Desastres ambientais: usar Grass para gerar possíveis zonas de inundação/incêndios florestais, com base em mapas com curvas de nível e mapas do terreno.

• Rotas

• Solução distribuída (PostGIS + PHP)

Preparar tabelas postgres e inserir percursos alternativos no Postgres

Preparar algoritmo, por exemplo, Dijkstra (php ou postgres)

Script php: Gerar percursos on-the-fly, a partir de critérios de inclusão/exclusão

• Solução Unificada - pgrouting

pgRouting é uma extensão do PostGIS. É uma solução completa cuja estrutura de dados e algoritmos operam apenas no PostGIS, ao contrário da solução distribuída que usa a estrutura do PostGIS e opera o algoritmo numa linguagem do servidor.

Conclusão

Espera-se que o leitor não se assuste com a enorme variedade de software que foi aqui apresentado, pois variedade significa também riqueza de escolha e se se considerar-mos o número de profissionais envolvidos em todos estes projectos, então, só nos resta reconhecer o trabalho, esforço e dedicação que toda a comunidade internacional mantêm. Software Livre em SIG é sem dúvida um excelente exemplo de combinação de diferentes linguagens de programação.

Referências e Ligações Importantes

OSGeo - http://www.osgeo.org/
FOSS4G - http://foss4g.org/static/index.html
IGEO - http://www.igeo.pt/

1. http://www.wikipedia.org/
2. http://trac.osgeo.org/proj/
http://www.igeoe.pt/utilitarios/coordenadas/trans.aspx
3. http://trac.osgeo.org/proj/
4. http://code.google.com/intl/pt-PT/apis/maps/documentation/javascript/v2/introduction.html
5. http://www.scribd.com/doc/456573/Artigo-Comparacao-SGBDs-Geo
http://www.esteio.com.br/downloads/pdf/775-0310-DOC-VALI-TXT.pdf
6. http://www.qgis.org
7. http://www.opengis.es
8. http://www.gvsig.org
9. http://www.openjump.org
10. http://www.grass.itc.it
11. http://www.mapserver.org
12. http://geoserver.org
13. http://openlayers.org
14. http://www.mapbender.org
15. http://mapfish.org
16. Asynchronous JavaScript and XML - http://en.wikipedia.org/wiki/Ajax_(programming)
17. http://trac.osgeo.org/osgeo4w/
18. http://www.w3.org
19. http://www.opengeospatial.org

A consulta às referências foi realizada em Julho de 2010.

SOBRE O AUTOR

Marco Filipe Vidal Afonso, com 32 anos, é Técnico de Informática na Câmara Municipal de Tavira. Estudou durante 3 anos no curso de Engenharia Informática na Universidade de Évora. É um entusiasta pelo software livre, tem experiência no desenvolvimento de websites, aplicações web e em Sistemas de Informação Geográfica na vertente web. Como hobbies, gosta de ler livros de filosofia e de jogos de estratégia. Também faz parte das seguintes equipas de desenvolvimento: Mapas do Concelho de Tavira (http://mapas.cm-tavira.pt) e Diplomatic Wars (http://www.diplomaticwars.com)

Marco Afonso

FLEX - Fast Lexical Analyser

Esta série de artigos introduz as ferramentas Unix flex e byacc através duma abordagem prática, a implementação duma calculadora simples. Os artigos assumem a familiaridade do leitor com C e C++, a qual será necessária para seguir os exemplos. Antes de prosseguir para a calculadora no entanto, importa explicar o que são e para que servem o flex e o byacc.

• Flex: o flex, da autoria de Vern Paxson, também conhecido como "fast lexical analyser" é um gerador de analisadores lexicais. Um analisador lexical é uma ferramenta que permite ler uma input stream e tomar acções quando partes dessa stream correspondem a padrões definidos pelo programador. Estes padrões, normalmente conhecidos como expressões regulares e usados em muitas outras aplicações (grep, Visual Studio, Notepad++, etc.) tal como suportados por várias linguagens (Java, Perl, PHP, Python, Ruby, Visual Basic 6, etc.) são fundamentais para o uso do flex e serão abordados com mais profundidade noutro artigo.

• Byacc: o berkeley yacc, da autoria de Robert Corbett, é um gerador de parsers. Um gerador de parsers permite criar uma aplicação que recebe tokens e caso estes tokens conformem com a gramática especificada pelo programador são tomadas acções. Estes conceitos serão abordados com mais profundidade ao longo dos artigos.

O flex e o byacc são duas das variantes das ferramentas que durante décadas foram o standard dum gerador de analisadores lexicais e dum gerador de parsers em sistemas Unix, o lex e o yacc da AT&T, desenvolvidos nos anos 70 nos laboratórios Bell por Mike Lesk e Eric Schmidt (lex) e Stephen C. Johnson (yacc). Outras variantes são GNU Bison, MKS lex e yacc e Abraxas lex e yacc. Apesar de existirem diferenças entre as variantes, elas são em muitos casos negligenciáveis na óptica do utilizador, sendo possível correr muito do código lex directamente em flex ou outra variante sem modificações, o mesmo se verificando com código yacc usado em byacc ou bison.

Ainda que tenham outros usos actualmente, historicamente geradores de analisadores lexicais e geradores de parsers foram concebidos para escrever compiladores - yacc é um acrónimo de "yet another compiler compiler" (o "Compiler Compiler" original foi escrito em 1960 por Tony Brooker) e o lex foi desenvolvido posteriormente ao yacc, destinado a ser utilizado em seu complemento - função para a qual são utilizados ainda hoje.

Vamos agora usar estas ferramentas para escrever uma calculadora simples que funcione a partir da linha de comandos. Neste primeiro artigo focar-nos-emos no flex, deixando o byacc para um segundo artigo e a versão final da calculadora para um terceiro. Começando pelo flex então, importa primeiro descrever a sua estrutura. Um ficheiro flex é constituído por 3 secções delimitadas pelo separador %%. A primeira e a terceira secção são utilizadas para inserir código do utilizador, escrito em C/C++, sendo a primeira secção usada para as definições e #includes e a terceira para as rotinas principais do utilizador. A segunda secção é utilizada para as regras do flex propriamente ditas.

```
%%
.|\n  putchar(*yytext);
%%
```

Este é um exemplo muito simples de um ficheiro flex, a primeira e terceira secção encontram-se vazias porque não introduzimos código nosso e a segunda contém uma única regra que se limita a fazer output do input que o programa recebe. Vamos analisar esta regra primeiro:

```
.|\n   putchar(*yytext);
```

Uma regra em flex corresponde a uma expressão regular e uma acção. A expressão regular é um padrão ao qual o analisador lexical vai tentar encontrar correspondência na input stream. Quando encontra toma a acção associada à regra, neste caso putchar(*yytext);. A acção é código C/C++ do utilizador, e pode ser uma instrução solitária como acima, ou um bloco do tipo {} ou %{ %} que ocupe várias linhas. O ".|\n" é uma expressão regular que basicamente aceita qualquer caractere, o ponto significa qualquer caractere excepto o \n e a barra significa "ou", logo aquela expressão regular pode-se ler como "qualquer caractere excepto o \n ou o \n", portanto, tudo. O yytext é um char *, que corresponde à string à qual a expressão regular fez "match". Guardando aquele ficheiro como ex1.l e correndo na shell os seguintes comandos,

```
flex ex1.l
gcc -o ex1 lex.yy.c -lfl
```

obtemos um executável, ex1, que se comporta como o comando cat corrido sem argumentos. Vejamos então o que se passa aqui: o comando flex ex1.l está a gerar o ficheiro lex.yy.c, um ficheiro que implementa em C os comportamentos que descrevemos em flex. Em seguida

corremos o gcc como normal, sendo necessário linkar a libfl para gerar o executável.

O que o Flex e, como veremos, o Byacc, fazem portanto é traduzir um conjunto de regras simples para código C (ou C++) que as implementa, limpando e facilitando o design e gerando na maior parte dos casos código mais eficiente do que um programador faria manualmente.

```
%%
.|\n  putchar(*yytext);
%%

int main()
{
    yylex();
}
```

O código acima é idêntico ao anterior e se compilado vai comportar-se da mesma forma. A diferença aqui reside na introdução da função main na terceira secção. Anteriormente o main não foi introduzido porque caso ele esteja omisso o flex linka-o automaticamente, mas importa mencionar que é possível escrevermos o nosso próprio main caso o desejemos. Neste caso em particular o main é igual ao que o flex usa, limitando-se a chamar a função yylex(), onde estão implementadas as regras. Caso escrevamos o nosso main devemos lembrar-nos de chamar essa função para além do nosso código ou nenhuma das regras vai executar.

Vamos agora considerar um caso mais complexo, o da nossa calculadora. O que é que precisamos que o flex faça neste caso? Primeiro é preciso perceber que o ficheiro flex, o scanner, é a rotina de baixo nível, enquanto que o ficheiro byacc, o parser, é a de alto nível. É o parser que controla o scanner e o chama quando necessita de novo input. A função do scanner é ler a input stream, processá-la e informar o parser do que leu. Portanto o que queremos que o nosso ficheiro flex faça é ler o input do utilizador e enviá-lo para o parser. No caso duma calculadora o input que nos interessa são números, portanto queremos que o scanner tenha uma regra deste género:

```
num         retornaNumero();
```

Em que num corresponde a uma expressão regular que faz match a números e retornaNumero() à acção que processa o número e o envia para o scanner. O código flex que implementa isto é o que se segue:

```
DIG         [0-9]
MANTISSA
({DIG}*\.{DIG}+)|({DIG}+\.)
EXPONENT    [eE][-+]?{DIG}+
```

```
INT         (0|[1-9]{DIG}*)
DOUBLE      {MANTISSA}{EXPONENT}?

%%

{INT}|{DOUBLE}    { yylval.d =
strtod(yytext, NULL); return NUMBER; }

%%
```

Estão aqui vários conceitos novos, especialmente se se não conhecer expressões regulares, portanto vamos abordar o problema lentamente. Antes de mais o que é um número, ou mais precisamente, que representações existem para um número? É importante definir isto para sabermos que expressão regular devemos utilizar para lhe fazer correspondência. Conhecendo C/C++ é fácil de reconhecer que 1, 15, 9.2e-5, 3.5, .2, 4., 6.e3 e .45E+14 são todas representações válidas de números, portanto a nossa expressão regular tem que saber reconhecer todas. Comecemos pelo mais simples, a expressão regular que corresponde a um dígito é [0-9]. Esta sintaxe com os parênteses rectos indica uma classe de caracteres e representa um intervalo de caracteres ao qual queremos fazer match. Se quisermos fazer match a uma letra qualquer por exemplo, podemos escrever [a-z] (ou [A-Za-z] se quisermos maiúsculas e minúsculas).

É possível criar classes de caracteres para qualquer conjunto que nos interesse fazer match. Por exemplo se quisermos criar uma regra para apanhar vogais podemos escrever [aeiouAEIOU] indicando um a um os caracteres que pertencem ao nosso intervalo. É preciso tomar cuidado com o hífen dentro duma classe de caracteres, num caso como [a-z] o hífen é açúcar sintáctico para representar os caracteres de 'a' a 'z' e não os caracteres 'a', '-' e 'z'. Se quiséssemos o segundo caso deveríamos escrever [-az], com o hífen no início. Existem muitas outras nuances como esta no uso de expressões regulares, mas não é objectivo deste artigo abordá-las. Expressões regulares são um tema suficientemente vasto para merecer o seu próprio artigo, e de facto, livros. Esta introdução limitar-se-á a explicar o significado das expressões regulares que usar sem entrar em maior detalhe.

Tendo agora uma noção do que são classes de caracteres importa perceber como fazer match a um número com vários algarismos. Por exemplo o número 259. Poder-se-ia pensar na expressão regular [0-9][0-9][0-9] e neste caso funcionaria, mas tem vários problemas. O primeiro e mais óbvio é que esta expressão regular só faz match a números com três algarismos, o segundo e menos grave é a repetição de código. A solução do segundo é simples, basta fazer [0-9]{3}, mas a solução do primeiro implica introduzir um novo conjunto de metacaracteres, '*', '+' e '?'.

O asterisco significa zero ou mais quando associado a uma

expressão regular, ou seja [0-9]* faz match a zero ou mais algarismos, portanto "", "1", "123" e "43538" são todos aceites por esta expressão regular. É importante mencionar que o asterisco "come" tudo o que consegue portanto um número com n algarismos consecutivos vai sempre ser aceite por [0-9]*, independentemente do tamanho de n.

O sinal de mais funciona de forma semelhante ao asterisco, diferindo no facto que significa um ou mais e não zero ou mais, portanto no caso da expressão regular [0-9]+ os números aceites têm de ter sempre pelo menos um algarismo. O ponto de interrogação significa zero ou um, indicando um elemento opcional. Podemos escrever [-+]?[0-9]+ para indicar um número com sinal opcional ("13", "-1" e "+324" são aceites por esta expressão regular).

Um número com n algarismos pode ser então aceite pela expressão regular [0-9]* (ou [0-9]+ caso n > 0) mas com esta expressão regular só serão aceites números naturais. Se quisermos aceitar reais na notação descrita anteriormente necessitaremos de uma expressão regular mais complexa (inteiros negativos e reais negativos serão tratados pelo parser). Antes de começarmos a escrever essa expressão regular no entanto, vamos introduzir uma funcionalidade do flex que vai tornar o nosso código mais legível. Uma expressão regular para apanhar endereços de email simples como [-a-zA-Z0-9._%+]+@[-a-zA-Z0-9.]+\.[a-zA-Z]{2,4} pode ser difícil de absorver de relance, mas é uma expressão que o leitor já deverá compreender parcialmente, e quando adquirir alguma prática, totalmente. No entanto esta expressão evidência a tendência das expressões regulares para se tornarem ilegíveis quando atingem um certo grau de complexidade. Para evitar isso o flex permite criar definições, DIG [0-9] define a expressão regular [0-9] como DIG, permitindo aceder ao seu valor escrevendo {DIG}, ou seja, é equivalente escrever {DIG} ou [0-9].

```
MANTISSA
({DIG}*\.{DIG}+)|({DIG}+\.)
```

A expressão acima também é uma definição (razão pela qual está na primeira secção) e define MANTISSA como sendo zero ou mais dígitos, seguidos de um ponto e de um ou mais dígitos ou um ou mais dígitos seguidos de um ponto. Existem dois pormenores que ainda não abordámos neste caso, os parênteses e o "\.". O "\." é uma sequência de escape para indicarmos que queremos mesmo o caractere ponto e não o metacaractere ponto que indica qualquer caractere. Os parênteses são metacaracteres que servem como habitual para agrupar itens.

```
EXPONENT     [eE][-+]?{DIG}+

INT          (0|[1-9]{DIG}*)
DOUBLE       {MANTISSA}{EXPONENT}?
```

Este conjunto de definições não introduz nada de novo, mas convém analisar cada definição para assentar conceitos. EXPONENT é definido como um "e" ou "E" seguido de um mais ou menos opcional e um ou mais dígitoos. Um INT é um zero ou um numero de um a nove seguido de zero ou mais dígitos. Um DOUBLE é uma MANTISSA seguida de um EXPONENT opcional. O comportamento do metacaractere "|", ou, é sempre como indicado na segunda definição, tudo o que vem antes do "|" ou tudo o que vem a seguir.

Tendo concluído a secção das definições vamos analisar as regras, ou a regra neste caso, visto só termos uma.

```
{INT}|{DOUBLE}   { yylval.d =
strtod(yytext, NULL); return NUMBER; }
```

A expressão regular neste caso é simples graças ao uso de definições, aceita um INT ou um DOUBLE. A acção também é simples mas a compreensão total envolve pormenores que só vamos ver quando começarmos a escrever o parser. Por enquanto basta saber que o yylval é uma union definida pelo parser onde guardamos o valor de retorno e o NUMBER é um #define que indica o tipo do retorno. O yylval.d é do tipo double, mas o que o flex fez match foi a uma string, portanto usamos a função strtod() da stdlib para converter a string para double antes de a enviarmos para o parser.

Agora que já temos o scanner a aceitar números, falta-nos aceitar as operações da calculadora e lidar com o restante input. Esta nova versão do ficheiro trata disso.

```
%option outfile="CalculatorScanner.c"

%{
#include "CalculatorParser.tab.h"


void yyerror(char * err_msg)
{
    printf("%s\n", err_msg);
    exit(1);
}

%}

WS          [ \t]
NL          "\n"|"\r"|"\r\n"
DIG         [0-9]


MANTISSA
({DIG}*\.{DIG}+)|({DIG}+\.)
EXPONENT    [eE][-+]?{DIG}+

INT         (0|[1-9]{DIG}*)
DOUBLE      {MANTISSA}{EXPONENT}?
```

```
%%

{INT}|{DOUBLE}    { yylval.d =
strtod(yytext, NULL); return NUMBER; }

[-+*/^()]         return *yytext;

{NL}              return NL;
{WS}              ; /* ignore
whitespace */

.                 yyerror("Unknown
character");

%%
```

Vejamos o que foi introduzido aqui. Na primeira linha temos a seguinte instrução,

```
%option outfile="CalculatorScanner.c"
```

cujo propósito é simplesmente indicar ao flex que o nome do ficheiro que queremos que ele gere é "CalculatorScanner.c" e não o default "lex.yy.c". Em seguida abrimos um bloco de código C com "%{", e fechamo-lo mais tarde com "%}". Podemos utilizar estes blocos para inserirmos código C/C++ como usual. Inserir código C/C++, incluindo comentários, fora destes blocos não vai funcionar como esperado a menos que todo esse código esteja indentado.

```
#include "CalculatorParser.tab.h"
```

Este #include - que também era necessário na versão anterior mas foi omitido por motivos de simplificação - é um header criado pelo byacc e que contém a union e os #defines mencionados anteriormente. A função yyerror() que se lhe segue, é uma função C normal, criada para lidar com erros no input. Veremos à frente como é usada.

```
WS          [ \t]
NL          "\n"|"\r"|"\r\n"
```

Estas duas definições fazem respectivamente match a espaço em branco (quer sejam espaços ou tabs) e a mudanças de linha (quer sejam de linux, mac ou windows). As aspas na definição NL são um metacaractere e indicam que queremos fazer match exactamente ao que está lá dentro. No exemplo da expressão regular para os reais, quando usámos a expressão \. para dizer que queríamos o caractere ponto, podíamos ter usado a expressão "." para o mesmo efeito, tal como aqui podíamos ter usado a expressão \\n|\\r|\\r\\n em vez da que usamos.

```
[-+*/^()]         return *yytext;
```

Esta classe de caracteres apanha todas as operações que a nossa calculadora vai realizar, somas, subtracções, multiplicações, divisões e potências, tal como parênteses para podermos criar agrupamentos. Neste caso retornamos o caractere directamente ao parser, algo que podemos fazer sempre que o input ao qual queremos fazer match é só um caractere.

```
{NL}              return NL;
```

No caso duma mudança de linha não existe nenhum valor que nos interesse retornar ao parser, portanto basta-nos informar o parser sobre o acontecimento. A necessidade de efectuar este retorno tornar-se-á mais óbvia quando discutirmos o parser.

```
{WS}              ; /* ignore
whitespace */
```

O espaço em branco não interessa à nossa calculadora, mas queremos permitir que o utilizador o possa usar, portanto usamos uma instrução vazia como acção, causando com que o scanner o ignore.

```
.                 yyerror("Unknown
character");
```

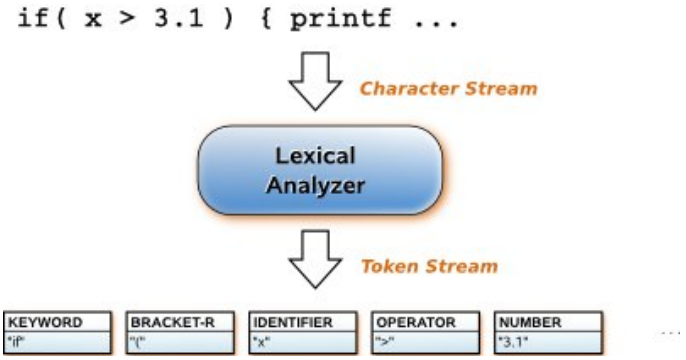
Esta última regra destina-se a lidar com input inválido. A primeira consideração a tomar é que esta regra necessita de estar em último lugar porque o ponto faz match a tudo. A forma como o flex funciona é a seguinte, dado input que pode ser aceite por duas regras, o flex usa a que aceita mais input. Se as duas aceitarem a mesma quantidade o flex usa a que vem primeiro no ficheiro. Tomemos o seguinte caso:

```
%%

[0-9]{3}      printf("número com 3
algarismos");
[0-9]*        printf("número com n
algarismos");

%%
```

Para o input "123456" o output do scanner vai ser "número com n algarismos" e não "número com 3 algarismosnúmero com 3 algarismos" como seria se ele usasse a primeira regra duas vezes. A razão disto é que a segunda regra aceita mais do input que a primeira, como tal neste caso, é essa que o flex usa. Para o input "123" no entanto, o output vai ser "número com 3 algarismos", já que, aceitando as duas regras

a mesma quantidade do input, o flex usa a que aparece primeiro, [0-9]{3}.

No caso do ponto, ele vai apanhar todo o input que as regras que o precedem não aceitem. Isto resulta porque o ponto só aceita um caractere individual. Se em vez de "." tivéssemos escrito ".*" ele usaria esta regra em detrimento de todas as outras (excepto as que contivessem \n) independentemente de onde a colocássemos no ficheiro. Percebendo em que casos é que esta regra é utilizada, basta decidir como queremos lidar com o input inválido. No nosso ficheiro estamos a lançar um erro e a parar a execução mas podíamos só ignorar o input inválido da mesma forma que os espaços, ou emitir só um aviso. Com isto concluímos o scanner para a nossa calculadora e este artigo. No próximo artigo veremos byacc, e como implementar o parser.



SOBRE O AUTOR

Estando actualmente no terceiro ano de LEIC no IST, interssa-se preferencialmente por arquitectura de computadores e computação gráfica.

João Mares

LUA - Linguagem de Programação - Parte 5

No artigo anterior demos enfâse às informações relacionadas com o uso de concatenação, precedências, variáveis indexadas, tabelas com listas, e registos. O tema deste artigo é o uso das funções.

A finalidade geral de uma função é o de retornar um valor após a execução de sua operação. Na linguagem de programação Lua, uma função poderá retornar um ou mais valores e até mesmo não retornar nenhum valor.

A linguagem Lua faz uso de funções internas e externas. Neste artigo será enfatizado o uso de funções externas.

Funções Internas

Funções internas relacionam-se ao conjunto de recursos existentes e disponíveis nas bibliotecas internas de uma linguagem.

Lua possui como conjunto de bibliotecas padrão, as bibliotecas:

- básica;
- de pacotes;
- de cadeias de caracteres;
- manipulação de tabelas;
- funções matemáticas;
- entrada e saída;
- facilidades do sistema operacional;
- facilidades de depuração.

Além das funções de bibliotecas padrão, há outras funções que poderão ser facilmente consultadas no site da linguagem Lua: www.lua.org/manual/5.1/pt/manual.html.

Nos artigos anteriores, em alguns dos exemplos de programas apresentados, foi utilizada a função **string.format()**, que se caracteriza por ser uma função da biblioteca de cadeias de caracteres.

O programa seguinte faz a apresentação de alguns efeitos relacionados à string "**Linguagem Lua**" definida para a variável **FRASE**. Serão apresentados o tamanho da frase (**string.len**), em caracteres minúsculos (**string.lower**), em caracteres maiúsculos (**string.upper**) e em caracteres invertidos (**string.reverse**).

```
-- início do programa FUNCAO 1

  FRASE = "Linguagem Lua"

  print(string.len(FRASE))
  print(string.lower(FRASE))
  print(string.upper(FRASE))
  print(string.reverse(FRASE))

-- fim do programa FUNCAO 1
```

De seguida, escreva o código do programa num editor de texto, gravando-o com o nome funcao01.lua e execute-o com a linha de comando lua 5.1 funcao01.lua.

Outra biblioteca padrão muito requisitada é a biblioteca de funções matemáticas onde se destacam funções para diversos cálculos matemáticos. O programa seguinte faz a apresentação de alguns cálculos matemáticos obtidos a partir de funções matemáticas, tais como: co-seno, seno, tangente, raiz quadrada e inteiro do valor 1.12.

```
-- início do programa FUNCAO 2

  VALOR = 1.12

  print(math.cos(VALOR))
  print(math.sin(VALOR))
  print(math.tan(VALOR))
  print(math.sqrt(VALOR))
  print(math.floor(VALOR))

-- fim do programa FUNCAO 2
```

De seguida, escreva o código num editor de texto, gravando-o com o nome funcao02.lua e execute-o com a linha de comando lua 5.1 funcao02.lua.

Funções Externas

Funções externas são um conjunto de funcionalidades que podem ser definidos pelo próprio programador. A linguagem Lua oferece alguns recursos operacionais muito atraentes no uso de funções definidas pelo programador.

Funções externas podem ser utilizadas com ou sem passagens de parâmetros e podem também ser usadas como comandos quando se desejar simular operações

equivalentes ao uso de sub-rotinas (procedimentos em linguagem PASCAL).

O programa seguinte apresenta o resultado do quadrado de um valor lido via teclado.

```
-- início do programa FUNCAO 3

  function QUADRADO()
    QUAD = VLR * VLR
    return QUAD
  end

  VLR = io.read("*number")
  print(QUADRADO())

-- fim do programa FUNCAO 3
```

Escreva o código num editor de texto, gravando-o com o nome funcao03.lua e execute-o com a linha de comando lua 5.1 funcao03.lua.

Em relação ao programa funcao03.lua, este após a entrada do valor na variável **VLR** faz a chamada da função **QUADRADO()** que por sua vez pega o valor atribuído à variável **VLR**, calcula o quadrado, armazena o seu resultado na variável **QUAD**, o qual é pela instrução return **QUAD** devolvido como resultado para a função **QUADRADO()**. Observe que a definição de uma função é feita com os comandos **function** e **end**.

No programa funcao03.lua há outro factor a ser considerado, sendo o uso das variáveis **VLR** e **QUAD** como variáveis globais.

As variáveis utilizadas na linguagem Lua caracterizam-se por poderem ser variáveis globais ou locais, além de serem também utilizadas como campos de tabela como mostrado no artigo anterior.

Uma variável global possui visibilidade que lhe confere a capacidade de ser vista por todo o programa, incluindo a parte principal e as funções. Quando uma variável é usada e definida num programa Lua, esta é por padrão considerada global, a menos que seja explicitamente definida como local por meio do comando local.

Por exemplo, ao olhar para o código do programa funcao03.lua nota-se que a variável **VLR** é citada tanto na parte principal do programa como na função. Desta forma, esta variável deverá ser global, mas a variável **QUAD** é usada apenas no código da função. Assim sendo, esta variável não tem a necessidade de ser global, podendo ser definida como variável local. Desta forma, observe o código seguinte.

```
-- início do programa FUNCAO 4

  function QUADRADO()
    local QUAD = VLR * VLR
    return QUAD
  end

  VLR = io.read("*number")
  print(QUADRADO())

-- fim do programa FUNCAO 4
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao04.lua e execute-o com a linha de comando lua 5.1 funcao04.lua.

Note que no programa funcao04.lua ocorre o uso do comando local antes da variável **QUAD**. Esta instrução fez com que **QUAD** seja uma variável local. Desta forma, ocorre um consumo menor de memória, pois após a conclusão da execução da função a variável **QUAD** é eliminada da memória.

Outra possibilidade de uso de funções em Lua é o uso de funções com passagem de parâmetros. Considere o seguinte programa que solicita a entrada de um valor numérico e apresenta como resultado o factorial do valor fornecido.

```
-- início do programa FUNCAO 5

    function FACTORIAL(N)
      local I, FAT
      FAT = 1
      for I = 1, N do
        FAT = FAT * I
      end
      return FAT
    end

  VLR = io.read("*number")
  print(FACTORIAL(VLR))

-- fim do programa FUNCAO 5
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao05.lua e execute-o com a linha de comando lua 5.1 funcao05.lua.

O programa funcao05.lua utiliza variáveis locais no código da função **FATORIAL(N)**, além do uso do parâmetro **N** que define o valor máximo de execução do ciclo **for**. Outro detalhe no código da função é o uso do limite do ciclo como sendo **I = 1, N** diferente de **I = 1, N, 1**. O uso do valor 1 após a definição de **N** é opcional, sendo obrigatório apenas quando o passo da contagem for maior que 1.

Um detalhe no uso de funções que deve ser considerado é em relação ao uso de return para que o valor calculado seja retornado. Este comando é de certa forma opcional, pois aquando da sua ausência, a função não dará retorno de um valor. Neste caso, a função terá um comportamento semelhante à definição de procedimentos na linguagem PASCAL.
Normalmente funções sem retorno exigem o uso de variáveis globais.

O programa seguinte apresenta o resultado do cálculo do factorial de um número por meio de uma função que não retorna valor.

```
-- início do programa FUNCAO 6

    function FACTORIAL(N)
      local I, FAT
      FAT = 1
      for I = 1, N do
        FAT = FAT * I
      end
      print(FAT)
    end

  VLR = io.read("*number")
  FACTORIAL(VLR)

-- fim do programa FUNCAO 6
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao06.lua e execute-o com a linha de comando lua 5.1 funcao06.lua. Observe junto ao programa funcao06.lua que o resultado da operação é impresso dentro do código da função.

Há uma forma de definição de função em Lua onde se associa a acção da função ao nome de uma variável. Essa forma de uso é também denominada função anónima.

O programa seguinte apresenta este modo de operação. Observe os detalhes de uso da linha de código **FACTORIAL = function(N)**.

```
-- início do programa FUNCAO 7
    FACTORIAL = function(N)
      local I, FAT
      FAT = 1
      for I = 1, N do
        FAT = FAT * I
      end
      return FAT
    end

  VLR = io.read("*number")
  print(FACTORIAL(VLR))
-- fim do programa FUNCAO 7
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao07.lua e execute-o com a linha de comando lua 5.1 funcao07.lua.

A definição de uma função assim como uma variável também poderá ser local. Portanto, observe o uso do comando local antes da definição da função no código seguinte.

```
-- início do programa FUNCAO 8

    local function FACTORIAL(N)
      local I, FAT
      FAT = 1
      for I = 1, N do
        FAT = FAT * I
      end
      return FAT
    end

  VLR = io.read("*number")
  print(FACTORIAL(VLR))

-- fim do programa FUNCAO 8
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao08.lua e execute-o com a linha de comando lua 5.1 funcao08.lua.

Parâmetros Arbitrários

Nos códigos de programas funcao05.lua e funcao06.lua foi utilizado o conceito de passagem de parâmetros que também podem ser referenciados como argumentos.

É importante saber que a linguagem Lua não opera com passagem de parâmetro por referência, apenas faz uso de passagem de parâmetro por valor. No entanto, o código de uma função escrito em Lua pode retornar um número arbitrário de parâmetros.

```
-- início do programa FUNCAO 9

   function QUANTIDADE(N)
     if (N == 1) then
       return "UM"
     end
     if (N == 2) then
       return "UM", "DOIS"
     end
     if (N == 3) then
       return "UM", "DOIS", "TRES"
     end
   end
```

```
  VLR = io.read("*number")
  print(QUANTIDADE(VLR))

-- fim do programa FUNCAO 9
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao09.lua e execute-o com a linha de comando lua 5.1 funcao09.lua.

Observe que ao ser executado o código do programa funcao08.lua o retorno dependerá do valor fornecido como parâmetro para a função **QUANTIDADE(N)**.

Outra forma de uso de arbitrariedade de parâmetros é quando os parâmetros a serem usados são indefinidos. Observe no código seguinte a definição de um parâmetro arbitrário por meio da definição de três pontos (...) que caracteriza-se por ser a definição de uma expressão do tipo *vararg*.

```
-- início do programa FUNCAO 10
    local function FACTORIAL(...)
      local I, FAT
      FAT = 1
      for I = 1, ... do
        FAT = FAT * I
      end
      return FAT
    end

  VLR = io.read("*number")
  print(FACTORIAL(VLR))
-- fim do programa FUNCAO 10
```

Escreva o código de programa num editor de texto, gravando-o com o nome funcao10.lua e execute-o com a linha de comando lua 5.1 funcao10.lua.

Uma expressão vararg pode ser utilizada quando definida dentro de uma função que possua um número variável de argumentos. Este tipo de operação não ajusta a sua lista de parâmetros. Ao invés disso, pega em todos os parâmetros extras, fornecendo-os à função em uso por meio de uma expressão *vararg*. O valor desta expressão é uma lista de todos os argumentos extras correntes, semelhante a uma função com retorno de múltiplos valores.

Recursividade

Não há como não falar de uso de funções sem considerar o efeito de recursividade. Neste sentido, a linguagem Lua também oferece tal recurso. Vale a pena lembrar que o efeito de recursividade é a capacidade que uma função possui de chamar a si própria para efectivar certo cálculo. O

```
-- início do programa FUNCAO 11
  local function FACTORIAL(N)
    if (N <= 1) then
      return 1
    else
      return FACTORIAL(N-1)*N
    end
  end

  VLR = io.read("*number")
  print(FACTORIAL(VLR))
-- fim do programa FUNCAO 11
```

programa seguinte demonstra o uso de função recursiva na linguagem Lua.
Escreva o código de programa num editor de texto, gravando-o com o nome funcao11.lua e execute-o com a linha de comando lua 5.1 funcao11.lua. Note junto ao programa funcao11.lua o uso da linha de instrução return onde encontra-se **FACTORIAL(N-1)*N** a chamada recursiva da função **FACTORIAL()**.

Conclusão

Neste artigo foi dada atenção às acções de processamento com uso de funções criadas com a linguagem de programação Lua. O conteúdo aqui exposto é introdutório, uma vez que o uso e criação de funções é um tema extenso e para o seu domínio é necessário tempo de dedicação e estudo.

No próximo artigo será tratado o assunto de uso de arquivos.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem experiência em ensino e desenvolvimento de programação de software desde 1986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

GOSTAS DE PROGRAMAÇÃO?

É DIFÍCIL ENCONTRARES TUTORIAIS EM PORTUGUÊS DE PROGRAMAÇÃO?

SENTES FALTA DE LER O QUE TE INTERESSA?

REVISTA

PROGRAMAR

acede já a

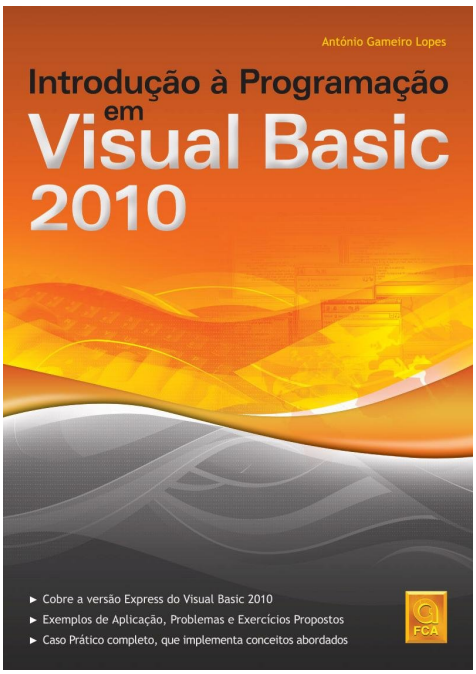
www.revista-programar.info

e vira uma página na tua vida

um projecto

portugal-a-programar.org

Introdução à Programação em Visual Basic 2010



O livro "Introdução à Programação em Visual Basic 2010", de autoria de António Gameiro Lopes, é um livro lançado pela FCA - Editora de Informática, com 448 páginas divididas em 8 capítulos, e é apresentado como "um guia de iniciação ao fascinante mundo da Programação, tomando como base o Visual Basic 2010, na sua versão gratuita Express"

Começa por definir os objectivos e organização do livro, assim como as convenções utilizadas durante este. De seguida, explica passo a passo a instalação do Visual Studio, com recurso a várias imagens, e a descrição o Ambiente de Desenvolvimento Integrado (IDE). O terceiro capítulo é dedicado a controlos e à construção da interface gráfica, descrevendo a Caixa de Ferramentas (toolbox), algumas noções fundamentais, propriedades, métodos, eventos e a descrição das classes mais utilizadas. O quarto capítulo, sobre noções básicas, descreve dados (constantes e variáveis), operadores, funções matemáticas, funções literais, como trabalhar com a data e hora e a gestão de ficheiros, pastas e drives. O quinto capítulo aborda estruturas, enumerações, variáveis indexadas (arrays), estruturas de decisão, estruturas de repetição, procedimentos, gestão de erros, etc. No sétimo capítulo, cujo objectivo é explicar outras funcionalidades do Visual Basic 2010, mostra como trabalhar com ficheiros ASCII (texto), como comunicar com o Microsoft Excel e a criação de elementos gráficos. O sétimo capítulo é um caso prático, onde mostra como se aplica alguns dos conhecimentos anteriormente referidos. Para finalizar, o oitavo e último capítulo, refere como efectuar a distribuição e publicação de uma aplicação.

«é um livro de leitura simples, agradável de seguir, bem organizado e que é adequado e recomendado a todos os que querem iniciar-se»

Como aspectos negativos pode-se apontar a não utilização da nova sintaxe do Visual Studio 2010/.NET Framework 4.0, como a continuação implícita de linha (a não utilização do underscore), embora este ainda possa ser utilizado. Utiliza também muito sintaxe que já têm substituto, e que em muitos casos existe por questões de compatibilidade entre versões, como o Msgbox(), CInt(), Len(), Left(), Right(), Ucase(), On Error Goto ..., etc. Embora que em alguns casos explique os novos métodos, seria interessante a utilização deles em todo o livro, pois ajudaria o leitor a uma melhor abordagem à "nova" sintaxe da linguagem.

Não aborda lamentavelmente conceitos base de Programação Orientada a Objectos (POO). A criação de classes, conceitos de herança, encapsulamento, polimorfismo e abstracção, seriam sem dúvida uma mais-valia neste livro.

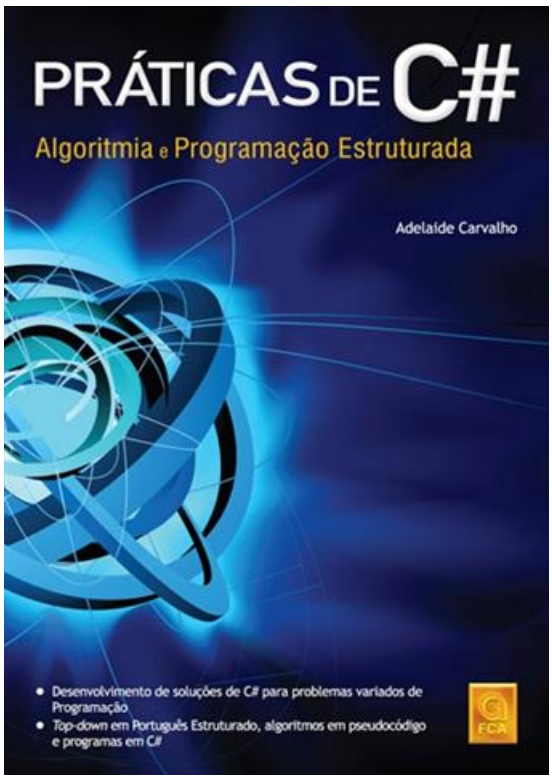
Por outro lado, como aspectos positivos, pode-se destacar a utilização de muitas fotos, tabelas de cursores, controlos, gráficos, etc, e pela descrição bem detalhada e clara em todos os capítulos. A utilização de fluxogramas para explicar fluxos de decisão, a abordagem a conceitos de recursividade e a comunicação com o Microsoft Excel, são sem dúvida uma boa aposta. Tem ainda bastantes exercícios e exemplos que ajudam o leitor a colocar em prática e a compreender os conceitos abordados.

No geral, e como conclusão, é um livro de leitura simples, agradável de seguir, bem organizado e que é adequado e recomendado a todos os que querem iniciar-se nesta linguagem de programação. Para os utilizadores que já têm alguns conhecimentos e que querem aprofundar um ou outro conceito, pode também ser uma boa opção de compra.

Sobre "Introdução à Programação em Visual Basic 2010"
ISBN 13: 978-972-722-644-3
P.V.P.: 27,75€
N.º de pág. 448

Jorge Paulino

Práticas de C#: Algoritmia e Programação Estruturada



O livro "Práticas de C#" foca todo o seu conteúdo na aprendizagem dos conceitos base de C#, usando programação estruturada. Ao longo do livro, são introduzidos conceitos desde operadores básicos, até a aspectos mais avançados como a validação, correcção e erro, e recorrência.

O primeiro capítulo é inteiramente dedicado à interacção do programador com o ambiente de desenvolvimento Microsoft Visual C# Express, disponibilizando assim uma introdução rápida das funcionalidades da ferramenta para quem ainda não teve contacto com a mesma. Os 5 capítulos seguintes focam os aspectos base da linguagem: variáveis e tipos de dados, programas sequenciais, estruturas de decisão e controlo, vectores e matrizes. Os restantes 3 capítulos falam sobre a validação e correcção de erros, funções e procedimentos, e conclui com as funções definidas por recorrência.

«... cada capítulo tem um conjunto de exercícios, perfazendo um total de mais de 180»

Fazendo jus ao seu nome, o livro "Práticas de C#" tem uma vertente muito prática: cada capítulo tem um conjunto de exercícios, perfazendo um total de mais de 180, a maioria deles bastante interessante. Todos estão resolvidos passo a passo (abordagem top-down ao problema, algoritmo em pseudo-código, e algoritmo em C#), para que seja possível aprender uma estratégia de resolução do problema, ou ainda comparar com outra solução desenvolvida. Todos os capítulos têm ainda uma explicação introdutória de cada tema, com pequenos exemplos, oferecendo as bases necessárias para acompanhar os problemas desse capítulo de forma rápida.

«É um livro aconselhado a quem pretende aprender C# e nunca tinha tido contacto com a programação»

É um livro aconselhado a quem pretende aprender C# e nunca tinha tido contacto com a programação, ou para quem já sabe os básicos da linguagem e pretende aprender mais sobre algoritmos, e de como resolver problemas de programação com C#.

Sobre "Práticas de C# - Algoritmia e Programação Estruturada"
ISBN 13: 978-972-722-638-2
P.V.P.: 27,50 €
N.º de pág. 368

André Silva

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