PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº6 - Janeiro de 2007 www.portugal-a-programar.org

Introdução à Programação em Scheme

Inicie-se na programação com esta excelente linguagem.

Iniciação ao Python

Aprenda a programar em Python.

Segurança: Exploits

Aprenda a dominar os Exploits para se proteger.

Gravação no Desktop

Guia completo para criar videos do seu desktop em linux

NOTÍCIAS: Retrospectiva sobre os principiais acontecimentos tecnológicos de 2006

índice

3 notícias
4 tema de capa
11 a programar
24 segurança
28 tecnologias
31 tutorial
37 gnu/linux
40 análises
41 internet
42 blue screen
43 comunidade

equipa PROGRAMAR

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

Um ano, seis edições

Esta sexta edição marca o final do primeiro ano da Revista PROGRAMAR. Para nós trata-se de um feito, para além de termos conseguido editar seis edições, o facto de o projecto ter crescido de edição para edição, com mais e melhores artigos, mais participantes e, principalmente, mais leitores. Muitas vezes, neste tipo de projecto, o mais difícil não é reunir um conjunto de artigos e formar uma revista, é conseguir manter o projecto activo e, se possível, em evolução, na direcção dos leitores.

Nesta edição contamos com um tema de capa mais extenso que o normal, vantagens de ser uma revista online, sem responsabilidades para com editoras. Esperamos que o tema seja do vosso agrado e, caso seja desconhecido, o que deverá acontecer ainda para muitos, que abram a mente e experimentem uma nova maneira de ver a programação, nem que seja pelos frutos que dela poderão extrair futuramente. Temos também uma nova secção, "Segurança", que surge na continuação do trabalho efectuado na terceira edição da revista.

Neste momento conseguimos atingir uma variedade considerável e muito apreciada dentro do nosso grupo de redactores e continuamos à procura de novos temas para abordar, novas secções para acrescentar e novas ideias para aplicar, e todos os novos membros que se juntam a nós trazem sempre novas opiniões ao grupo.

Sérgio Santos

Os principiais acontecimentos tecnológicos de 2006

2006 chegou ao fim e a revista Programar recorda as novidades mais importantes do ano que agora terminou.

O ano começou com uma revolução na Apple. Depois de uma década a usar PowerPC, a empresa de Steve Jobs resolveu mudar para a arquitectura da Intel. Apesar do desempenho dos novos macs ter melhorado significativamente, este primeiro ano fica marcado por alguns problemas de sobreaquecimento do processador. Em Abril a Apple apresentou também o Boot Camp, um software que permitiu aos novos computadores correrem nativamente Windows.

Na área dos processadores foi também um ano agitado, inicialmente com a AMD a conseguir ameaçar o domínio da Intel, sobretudo nos servidores com os Opteron. A Intel respondeu com o lançamento dos Core 2 Duo, que lhe permitiu regressar à liderança no poder de processamento. Tivemos também a passagem da tecnologia de 90 nm para 65, primeiro por parte da Intel, e no fim do ano pela AMD.

No que diz respeito aos browsers, surgiram igualmente novidades, primeiro com o lançamento do IE 7 da Microsoft, depois com a Fundação Mozilla a lançar o Firefox 2.0. Estas novas versões trouxeram várias novidades, das quais se destaca o filtro anti-phishing, denotando a cada vez maior preocupação com a segurança por parte dos produtores de software.

Depois de muitas promessas, a Sun finalmente abriu o código do Java. As plataformas J2SE e J2ME encontram-se agora sob a licença GPL, acessíveis a todos.

Depois de sucessivos adiamentos, o Vista foi finalmente lançado. Com um visual muito atractivo, o novo SO da Microsoft trouxe também novidades a nível da sua organização, do sistema de ficheiros e da segurança. Outra novidade foi a capacidade de se adaptar ao hardware da máquina em que é instalado, permitindo desta forma que também possa ser usado em PC's com menores recursos de hardware.

Em 2006 assistimos ao lançamento de duas novas consolas, a Wii e a PS3. Tal como seria de esperar, a vendas da PS3 só não foram maiores por falta de consolas nas lojas. Com um processador revolucionário e com excelentes gráficos, o único problema foi mesmo o elevado preço. Já a Wii apostou numa maior jogabilidade, com o seu comando inovador, o Wii-mote. Tal facto, juntamente com o preço acessível da consola contribuiu para o seu enorme sucesso.

2006 foi igualmente o ano da afirmação da web, com o enorme sucesso de sites como o YouTube, o MySpace, a Wikipedia, etc. e a proliferação dos blogs. A confirmar isto veio a escolha da revista Time para a personalidade do ano: todos aqueles que contribuiram para esta nova sociedade do conhecimento.

Scheme

Scheme é uma linguagem de programação que suporta o uso de múltiplos paradigmas, em particular, a programação imperativa e funcional.

Foi desenvolvida, originalmente, por Guy L. Steele e por Gerald Jay Sussman por volta dos anos 70. É uma linguagem que deriva do Lisp e, portanto, o seu dialecto e a forma sintáctica são idênticas. A utilização da programação funcional, em vez da programação imperativa (destruição de variáveis), torna a sua aprendizagem mais clara, uma vez que nada é modificado. Recebem-se objectos e retornam-se objectos.

Por ser uma linguagem bastante intuitiva e de simples compreensão, é possível aprender-se a programar em Scheme como primeira linguagem de programação. É uma linguagem muito minimalista quanto às suas funções de base. No entanto, graças à criação de bibliotecas e dependendo do que se quer fazer ou explorar na linguagem, é possível criarem-se programas, aplicações web, jogos, ou outro tipo de software como em qualquer outra linguagem de alto nível nomeadamente C, C++ ou Java.

Como em qualquer linguagem de programação, em Scheme existem vários tipos de dados. Estes vão desde expressões atómicas, a números símbolos e valores booleanos, a listas e a vectores. Exemplos:

```
(define a 5)
(define b #t)
(define c '(1 2 3))
(define d (+ 1 2 3 a))
```

Ligado ao primitivismo da linguagem, existe o conceito do cálculo das funções lambda. Que não são mais do que um procedimento anónimo. Este conceito é muito importante para associar nomes a procedimentos. Um exemplo:

```
(lambda (x) (* 2 x))
```

Esta expressão lambda, se executada, devolveria um procedimento mas que não fazia nada pois não lhe é passado um argumento de input à variável 'x'. No entanto, se fizéssemos:

```
> ( (lambda (x) (* 2 x)) 5)
10
```

Obteríamos o valor 10 que é o valor obtido ao passar o argumento de input 5 ao 'x' e de o procedimento anónimo lambda o operar consoante o seu corpo, onde, neste caso é multiplicar o 'x' por 2. No entanto, seria mais interessante poder chamar um nome a este procedimento para não estarmos constantemente a invocar este lambda anónimo de cada vez que queremos duplicar um real 'x'. Para isso fazemos:

```
(define duplica
    (lambda (x) (* 2 x)))
```

E para o executarmos fazemos:

```
> (duplica 5)
10
```

Uma vez que o nome da função "duplica" está associado a um procedimento lambda que por sua vez está à espera de um valor 'x' para o operar no seu corpo.

Se bem repararam, o dialecto do Scheme pode parecer um tanto estranho uma vez que estamos habituados a escrever, em matemática, os operadores no meio dos operandos: (5 + 3).

E portanto, um outro ponto forte do Scheme, que já vem do Lisp, é o facto de minimizar a ambiguidade da criação de funções e da sua avaliação. Por exemplo, em matemática temos enumeras maneiras de representar uma função, seja ela uma função factorial n!, polinomial x2+3x, ou mesmo uma função genérica f(x,y,z). Contudo, ainda há mais confusão porque existem prioridades de certas operações, como a prioridade da multiplicação perante a soma. O Scheme torna a coisa bastante mais simples e portanto tem a sua sintaxe da seguinte forma (função argumentos*). No início da expressão a ser avaliada vem o nome da função (operador) e seguido desse nome vem um conjunto de argumentos a serem operados (operandos).

Exemplos:

```
(* 2 x)
(+ 5 3)
(* 2 (+ 5 3) 9)
```

E utilizando a associação de expressões a nomes, poderíamos associar qualquer uma expressão das acima mencionadas a um nome que se tornava, por assim dizer, uma variável.

Demonstrando, um pouco melhor, o poder da programação funcional, fica o exemplo:

```
(lambda (+ *) (+ 5 (* 2 3) 4))
```

Este procedimento anónimo fica à espera da passagem de dois argumentos cujos nomes são '+' e '*' e há a tentação de os

associar às operações primitivas de somar e multiplicar respectivamente. Para se perceber, passemos aos 2 argumentos do lambda, dois argumentos de input:

```
> ((lambda (+ *) (+ 5 (* 2 3) 4))* +)
100
```

Uma vez que trocámos os nomes das operações que vão ser chamados no corpo do lambda, as operações de somar passam a ser multiplicações e as operações de multiplicar passam a ser somas. Como vêem, o poder de associação de nomes e de passagem de argumentos no Scheme é muito interessante e versátil.

Uma vez mais, como em qualquer linguagem que se preze, em Scheme existem expressões condicionais. A utilização de valores e expressões lógicas para condicionarmos algoritmos, utilizando os elementos lógicos >, <, =, >=, <=, not, and e or.

São caso disso o "if" e o "cond". A sintaxe do "if" é:

```
(if (condição)
    (expressão1 se condição é #t)
    (expressão2 se condição é #f))
```

Exemplo:

```
(define a 5)
(define b (+ a 1))

> (if(> a b)
      a
      b)
6
```

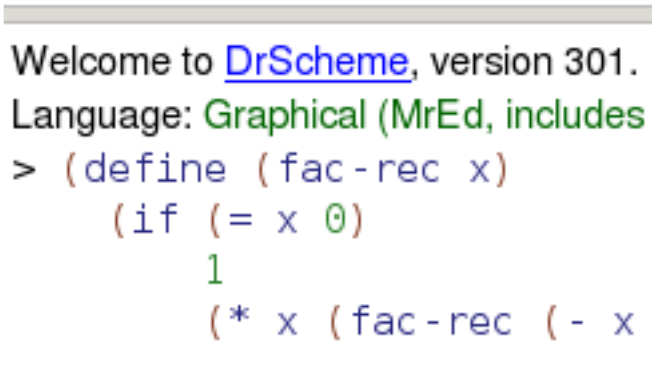
Muito simples. No entanto, e se quiséssemos complicar a coisa, poderíamos pôr como expressão de #t ou de #f, um outro if encadeado e portanto teríamos, por exemplo:

```
> (if(> a b)
      a
      (if(= b (+ a 1))
         a
         b))
5
```

Já se quiséssemos encadear novo if nas sub-expressões do segundo if, o código começava a tornar-se um pouco confuso de perceber. Para isso, utilizamos o acima mencionado "cond" que é, portanto, "açúcar sintáctico" para um conjunto de ifs encadeados como no exemplo em cima.
O cond recebe um conjunto de pares condição / expressão-if-#t e avalia-las por ordem. Se a primeira der #t, o seu corpo é executado, caso contrário passa para a seguinte e por aí em diante, até chegar a um else / expressão, caso nenhuma das expressões anteriores seja verdadeira. Conversão da expressão do exemplo anterior dos ifs encadedos para o cond:

```
> (cond((> a b)a)
       ((= b (+ a 1))a)
       (else b))
5
```

Como vêem é muito simples e dá muito jeito quando queremos encadear várias sucessões de condições.



A avaliação e interpretação de expressões em Scheme pode parecer confusa por não estarmos habituados à notação no entanto ficam aqui alguns exemplos que penso serem bastante elucidativos:

```
> (+ 2 3)
5

> (- 7 (+ 2 3) 2)
0

>(* (+ (- 7 1) 3) 4)
36

> (> 3 4)
#f

> (= 5 6)
#f

> (>= 4 3)
#t

> (and (> 4 3)
       (not (< 4 3)))
#t
```

Contudo, para aprendermos o que é programar, teremos de ter noção de "o que é um algoritmo?". Um algoritmo é um conjunto de operações finitas, executadas de forma mecânica num espaço de tempo finito de modo a realizarem uma determinada tarefa ou problema.

Por exemplo, se quiséssemos criar um algoritmo para calcular o factorial de um número em matemática tínhamos em mente que:

$0! = 1$
$n! = n(n-1)!$

Passando isto para Scheme, uma forma simples de o fazer seria:

```
(define (factorial n)
    (if (= n 0)
        1
        (* n (factorial ( - n 1)))))
```

Isto seria uma forma recursiva de criar um algoritmo que calcula o factorial de um inteiro natural.

Exemplo de interacção com a função usada:

```
> (factorial 4)
(* 4 (factorial 3))
(* 4 (* 3 (factorial 2)))
(* 4 (* 3 (* 2 (factorial 1))))
(* 4 (* 3 (* 2 (* 1 (factorial
0))))) => 1
(* 4 (* 3 (* 2 (* 1 1))))
(* 4 (* 3 (* 2 1)))
(* 4 (* 3 2))
(* 4 6)
24
```

Se repararem, existe uma fase de expansão e depois uma fase de contracção. Isto deve-se ao facto de este algoritmo do factorial gerar um processo recursivo e ser criado à custa de um procedimento recursivo. Embora pareçam a mesma coisa, não o são. Um processo recursivo tem como característica deixar operações pendentes no decorrer do programa e como reflexo disso, consumir memória ao longo do tempo. Um procedimento recursivo é o acto de uma função se invocar dentro de si própria "(factorial (- n 1))".

Para resolvermos o problema de deixarmos operações pendentes, poderíamos melhorar este algoritmo gerando um processo iterativo.

```
(define (factorial n)
  (define (fact-aux n acumulador)
    (if(= n 0)
      acumulador
      (fact-aux (- n 1)
                (* acumulador n))))
  (fact-aux n 1))
```

À primeira vista, isto pode gerar confusão mas vejamos por partes. Primeiro, foi introduzido o conceito de função interna. A função "fact-aux" só é visível à função que a precede hierarquicamente, a função "factorial", ou seja, ao nível global do ambiente, ela não existe.

Ao criarmos este algoritmo que gera um processo iterativo, este tem como característica não deixar operações pendentes e logo, de consumir memória constante, onde por sua vez, actualiza uma variável de estado que ditará o resultado final. Ou seja, a cada iteração, o valor do acumulador é actualizado com o produto do 'n' actual da iteração em que se vai pelo valor passado ao acumulador na iteração anterior. Se repararem, a primeira iteração é chamada com o 'n' superior (da função factorial original) e com o acumulador em 1. Isto porque o 1 é o elemento neutro da multiplicação e assim, não "destrói" o valor n.

Poderíamos, ainda, criar um algoritmo parecido com o factorial de um inteiro, mas que em vez de multiplicar os 'n' termos até 0, os soma. Temos então:

```
(define (soma-ate-zero n)
    (define (soma-aux n acumulador)
        (if(= n 0)
           acumulador
           (soma-aux (- n 1) (+ n
acumulador))))
    (soma-aux n 0))
```

E ao invocarmos:

```
> (soma-ate-zero 5)
15
```

Uma vez, mais, o acumulador é actualizado de iteração em iteração com a soma do acumulador passado da iteração anterior com o n da actual iteração.
Desta vez, o valor do acumulador passado à função interna na primeira iteração é o 0 (zero), visto ser o elemento neutro da adição.

Entrando noutra área, poderíamos abstrair-nos do conceito de factorial e de soma-ate-zero e fazermos um procedimento que recebe uma operação que opera inteiros e que calcula esta operação até ao inteiro 0.

```
(define(opera-ate-zero op elem-neutro n)
  (define (opera-aux n acumulador)
    (if(= n 0)
      acumulador
      (opera-aux (- n 1) (op n acumulador))))
  (opera n elem-neutro))
```

Assim, poderíamos definir o nosso factorial da seguinte maneira:

```
(define (factorial n)
    (opera-ate-zero * 1 n))
```

Ou seja, factorial n tem o elemento neutro 1 e a operação que é realizada até chegar ao valor 0 é o produto '*'. A isto chamamos abstracção de dados e procedimentos de ordem superior.

Para definirmos o soma-ate-zero em função do opera-ate-zero generalizado, faríamos:

```
(define (soma-ate-zero n)
    (opera-ate-zero + 0 n))
```

Para a subtracção:

```
(define (subtrai-ate-zero n)
    (opera-ate-zero – 0 n))
```

E para a divisão:

```
(define (divide-ate-zero n)
    (opera-ate-zero / 1 n))
```

Penso que fica aqui uma simples explicação e o conceito de operações entre procedimentos de ordem superior e a criação de funções para nos abstrairmos o máximo possível da função geral. A criação e utilização destes métodos de abstracção torna o código mais geral e reutilizável reflectindo-se num código com menos bugs e na não necessidade de constante manutenção.

Outro tipo de dados que vem no Scheme e que é muito importante e útil: o par. É o par que vai originar as listas. Em Scheme, um par cria-se da seguinte maneira:

```
> (cons elem1 elem2)
(elem1 . elem2)
```

Em que recebe dois argumentos e são associados como um par. Se pensarmos um bocadinho, em vez de passarmos como segundo argumento, um número, porquê não passarmos outro par para encadear? Aí teríamos, por exemplo:

```
> (cons 1 (cons 2 3)) elem
(1 . (2 . 3))
```

E isto é o princípio da criação de listas. No entanto, existem duas funções básicas quando se trabalha com o tipo de dados par. O "car" e o "cdr". O car e o cdr recebem um par como argumento e retornam o primeiro elemento e o segundo elemento do par, respectivamente. Ou seja, se fizermos:

```
> (car (cons 1 2))
1
> (cdr (cons 1 2))
2
> (car (cdr (cons 1 (cons 2 3))))
2
```

O Scheme suporta, ainda, um encadeamento destas formas sintácticas car e cdr, podendo fazer o seguinte:

```
> (cadr (cons 1 (cons 2 3)))
2
```

O Scheme aceita esta combinação de "c 'a's e 'd's r" até 5 vezes.

Posto isto podemos definir o tipo de dados listas. Uma lista não é mais do que um conjunto de pares encadeados que termina com o par (cons elem_n ()), em que "()" representa a lista vazia. Definamos, então, o tipo de dados lista:

Os construtores:

```
(define nova-lista ()) ;a lista vazia ()

(define (insere elemento LISTA)
    (cons elemento LISTA))
```

Se repararem, a função "insere" recebe com argumentos um elemento a inserir e a lista onde se vai inserir. Depois, no seu corpo, é feito o par (elemento. (LISTA)). No entanto, o cons já recebe dois argumentos para fazer o par, por isso poderíamos fazer esta função simplesmente assim:

```
(define insere cons)
```

Os selectores:

```
(define (primeiro-lista lista)
    (car lista))

(define (resto-lista lista)
    (cdr lista))
```

Se repararem, as funções primeiro-lista e resto-lista, recebem, ambas, listas para depois devolverem a respectiva parte, o primeiro elemento e o resto respectivamente. No entanto, o car e o cdr já recebem uma "lista" (um par), por isso podemos escrevê-las simplesmente assim:

```
(define primeiro-lista car)
(define resto-lista cdr)
```

Um reconhecedor:

```
(define (lista? X)
    (if(null? X)
        #t
        (if (pair? X)
            (lista? (resto-lista X))
             #f)))
```

Se o universal 'X' passado, for uma lista vazia, será, então uma lista, devolvendo #t. Se não for vazia mas for um par, é meio caminho andado para ser uma lista, ou seja, se sim ele vai ver se é lista o resto do universal 'X' que recebeu. Se todo o universal 'X' passar nos testes, devolve #t, devido ao pair? já devolver um #t ou #f, caso contrário devolve #f. Um teste:

```
(define (lista-vazia? LISTA)
    (if (null? LISTA)
         #t
         #f))
```

Podemos frisar dois aspectos importantes na minimização desta função. Da mesma maneira que nas funções anteriores, a função "lista-vazia?" recebe uma lista como argumento e depois vai avaliar se ela é vazia ou não através do operador "null?" que vem no Scheme. No entanto, o null? já recebe como argumentos, uma lista, pelo que poderíamos apenas escrever:

```
(define lista-vazia null?)
```

Outro aspecto na função primeiramente escrita é o de que o if mais os seus elementos de retorno "#t" e "#f", não precisavam de lá estar, uma vez que o null? já retorna um valor booleano deste tipo. Criado o nosso tipo de dados, define-se a chamada "barreira de abstracção de dados". Ou seja, a partir do momento que temos feita a axiomatização do nosso tipo de dados, utilizamos apenas as funções criadas para originar novos elementos (construtores), aceder-lhes (selectores), reconhece-los (reconhecedores) e avaliadores (testes).

Após a criação desta barreira de abstracção, uma função útil de criar seria a que nos dá o tamanho de uma lista:

```
(define (tamanho-lista LISTA)
  (if(lista-vazia? LISTA)
    0
    (+ 1 (tamanho-lista (resto-lista
                                LISTA)))))
```

Uma outra maneira de criar uma lista em Scheme é usar a função primitiva quote que é o mesmo que usar o carácter " ' " (plica).

```
> '(1 2 3)
(1 2 3)
> '((1 2) (2 3) (4 5))
((1 2) (2 3) (4 5))
```

Por exemplo, pondo em prática as funções já criadas para o tipo de dados listas, ficam aqui as seguintes interacções:

```
(define lista-vazia nova-lista)

(define lista2
    (insere 3 lista-vazia))
```

```
(define lista-final
    (insere 5 lista2))
```

Se mandássemos correr o seguinte comando:

```
> lista-final
(5 3)
```

Esta era a lista retornada. E poderíamos, então, correr a função que nos dá o tamanho dela:

```
> (tamanho-lista lista-final)
2
```

Ou mesmo o verificador de listas:

```
> (lista? lista-final)
#t
```

Já definimos uma função que insere um elemento à cabeça da lista. Dá jeito mas é bastante fraca. Se quisermos inserir um elemento numa dada posição faríamos a seguinte função:

```
(define (insere-pos elem pos LISTA)
(if(= pos 1)
(insere elem LISTA)
(insere-pos elem (- pos 1)
(resto-lista LISTA)))
```

Assim, se quiséssemos pôr o elemento 4 na "lista-final", criada em cima, na última posição, faríamos:

```
> (insere-pos 4 3 lista-final)
(2 3 4)
```

Acontece que poderíamos criar condições de erro para o utilizador, por exemplo para o caso da posição dada ser maior que o tamanho da própria lista, mas são aspectos pouco relevantes para o caso.

O tipo de dados lista, é muito fácil de usar e o Scheme nativo já inclui, por defeito, todas as funções aqui definidas e muitas mais. Em C, a criação de uma lista é muito mais complexa, uma vez que exige a criação de uma estrutura com um elemento e um ponteiro para a próxima estrutura. Em Scheme, isso já vem feito para nós. Outro tipo de dados parecido com as listas é o vector. É mais recorrente usar vectores em programação imperativa, chamando as posições com índices.

Virando o assunto para o paradigma do Scheme, a programação imperativa. A programação imperativa, baseia-se na destruição de variáveis, ou do seu conteúdo. Noutras linguagens como o C, é muito usual recorrer-se a este tipo de programação, já o Scheme tem o seu ponto forte na programação funcional, como foi referido no início.

No entanto, o Scheme também suporta este estilo de programação e para isso usa uma representação interna para alocar um valor e depois associa-o a um nome que por sua vez o associa a uma box. Uma box não é mais do que uma estrutura com um nome e com um valor lá dentro. Se invocarmos o nome da box, o Scheme interpreta o nome e faz o chamado unbox devolvendo o valor interno da box associado ao nome.

Posto isto, é possível perceber o que faz o operador set!. Esta função recebe um nome que tem obrigatoriamente de já ter sido definido com a função define e um novo valor ou expressão que retorne algo válido para o tipo de dados associado ao nome, e faz então a acção de unbox do nome, insere o novo valor (depois de avaliar a expressão seguida do nome) e volta a fazer box. É óbvio que isto é o que se sucede internamente e é o que diz a teoria do Scheme. Na verdade, tudo o que foi feito até agora respeita as regras teóricas do conceito Scheme enquanto linguagem de programação, porque no fundo, cada interpretador da linguagem e seu compilador podem gerir todos estes processos, internamente, de maneira diferente. Há que perceber a diferença entre o conceito de linguagem e sua sintaxe, e o que na verdade o seu interpretador faz.

Alguns exemplos da utilização do operador set!:

```
(define a 4)      (set! a 2)
(define b 2)      (set! b 0)
> a               > a
4                 2
> b               > b
2                 0
> (+ a b)         > (+ a b)
6                 2
```

Voltemos à função factorial que penso que elucida bem a diferença entre programação funcional e a programação imperativa:

```
(define(factorial-imp n)
  (define valor-final 1)
; inicialização com 1
; -> elemento neutro da multiplicação
  (define(aux n)
    (if(= n 0)
       valor-final
       (begin
         (set! valor-final (* n
                              valor-final))
         (aux (- n 1)))))
  (aux n))
```

É uma opção possível mas um pouco mais trabalhosa e que envolve destruição de valores. Passagem de argumentos entre funções sem as alterar é muito mais seguro e obtemos a mesma eficiência (nota: comparar com a versão funcional acima). Foi inserido o elemento begin. Em Scheme, só podemos realizar uma operação por corpo. Operação essa que devolva algo.

No entanto, existe uma função que recebe várias funções e que avalia todas e só devolve o valor retornado pela avaliação da última. Ou seja, no caso da função "factorial-imp", dentro da função auxiliar "aux", enquanto o "n" não chegar a 0 (zero), ele realiza o else do if, que corresponde a executar o begin. Nesse begin são avaliadas as expressões (set! valor-final (* n valor-final)) e (aux (- n 1))))), e só o valor da última é retornada. O acto de fazer set! não retorna nada. Minto, retorna void, que é interpretado como nada. Ou seja, é avaliado o set!, o valor de "valor-final" é destruído e renovado com outro e o factorial é invocado de novo recursivamente. Simplesmente, em vez de passarmos variáveis de estado actualizadas, criamos um acumulador que é actualizado usando o set! internamente.

Penso que a maior parte dos conteúdos acima explicados englobam e reflectem o poder do Scheme e a facilidade com que se pode aprender a programar usando esta linguagem. Foram apenas expostos exemplos simples mas que mesmo assim mostram o que é possível fazer em Scheme.

Para saber mais sobre esta linguagem aconselho um livro: Programação em Scheme – Introdução à programação utilizando múltiplos paradigmas – IST Press. Ou outras fontes como o do compilador e ambiente de interpretação Scheme http://www.plt-scheme.org. Este interpretador de Scheme é bastante poderoso em termos de opções e tem um modo de debug muito interessante, onde podemos aceder aos valores de todos os valores associados aos nomes presentes no código, simplesmente passando o rato por cima deles. Tem debug por steps, usa breakpoints e ainda tem disponível, para além do modo gráfico, um modo de consola. O programa está disponível para Linux, Windows, Mac, Sun, etc... É ainda possível criar executáveis dos programas criados para correr nas várias plataformas onde se estiver a trabalhar o código.

Espero que tenham percebido o potencial do Scheme como linguagem de alto nível mas, também, como linguagem de aprendizagem para se evoluir para outras linguagens.

Miguel Wahnon

Sockets em Java

Neste artigo vamos ficar a conhecer o suporte que Java oferece para a utilização do mecanismo de comunicação Socket, o mecanismo mais utilizado para a comunicação entre aplicações.

Java permite o uso de socktes pelos seguintes modos utilização:
-> Modo Orientado à Conexão:
Funciona com o protocolo TCP;
-> Modo Orientado ao Datagrama:
Funciona com o protocolo UDP.

Ambos os modos funcionam sobre o protocolo IP (Internet Protocol). Cada um destes modos tem a sua utilidade, vantagens e desvantagens na sua utilização.

Modo Orientado à Conexão (TPC/IP)

-> Vantagens:
• Serviços confiáveis, sem perda de dados na rede e ordem dos pacotes;
• Possibilidade de usar DataStreams.

-> Desvantagens:
• É mais lento que o modo orientado ao datagrama.
• O comportamento do servidor é diferente do comportamento do cliente.

Modo Orientado ao Datagrama (UDP/IP)

-> Vantagens:
• É bastante mais rápido que o modo orientado a conexão.
-> Desvantagens:
• Serviços não confiáveis, mensagens perdidas na rede e perda da ordem das mensagens;
• Cada mensagem é um datagrama: [Remetente, Destinatário, Conteúdo.].

Devido a uma maior utilização e estabilidade iremos apenas analisar e implementar o Modo Orientado à Conexão - TCP/IP.

Sockets TCP/IP

No processo de comunicação entre sockets TCP/IP, de uma forma simples, o servidor escolhe uma porta e aguarda conexões a essa porta, o cliente deve conter as seguintes informações:
• Endereço do Servidor (HOST);
• A porta usada pelo servidor (PORT).

Com essa informação o cliente solicita uma conexão ao servidor (Figura 1).

Se após o pedido de conexão não ocorrer nenhum problema, o servidor aceita a conexão gerando um socket numa porta do servidor, o que vai criar um canal de comunicação entre o cliente e o servidor (Figura 2).

Figura 1

Figura 2

Por norma o servidor funciona em ciclo (loop) esperando por novas conexões e criando sockets para solicitações de clientes.

Em seguida iremos ver as acções necessárias para implementar comunicações sobre TCP através de um socket cliente e um socket servidor.

Socket Client

1 – Abrir Conexão.

```
import java.io.*;
import java.net.*;
//Conectar ao servidor localhost na porta 8080.
Socket client = new Socket("127.0.0.1",8080);
```

2 – Obter Streams de entrada e saída para comunicação com o servidor.

```
//Cria um canal para receber dados.
DataInputStream in = new DataInputStream(client.getInputStream());
//Cria um canal para enviar dados.
DataOutputStream out = new DataOutputStream(client.getOutputStream());
```

3 – Realizar a comunicação com o servidor.

```
//Envia o inteiro 3000.
out.writeInt(3000);
//Envia a String “Olá - Socket Cliente.”.
out.writeUTF("Olá - Socket Cliente.");
//Espera pela recepção de um inteiro.
int valor = in.readInt();
//Espera pela recepção de uma String.
String texto = in.readUTF();
```

4 – Fechar as Streams e a conexão.

```
//Fecha o canal de entrada.
in.close();
//Fecha o canal de saída.
out.close();
//Fecha o Socket.
client.close();
```

Socket Servidor

1 – Criar Socket Server

```
import java.io.*;
import java.net.*;
//Cria um socket servidor na porta 8080.
ServerSocket serveer = new ServerSocket(8080);
```

2 – Aguardar Conexões

```
//O método accept retorna um socket para comunicação com o próximo cliente.
Socket sock = server.accept();
```

3 – Obter Streams de entrada e saída para comunicação com o cliente

```
//Cria um canal para receber dados.
DataInputStream in = new DataInputStream(sock.getInputStream());
//Cria um canal para enviar dados.
DataOutputStream out = new DataOutputStream(sock.getOutputStream());
```

4 – Realizar a comunicação com o cliente

```
//Espera pela recepção de um inteiro.
int valor = in.readInt();
//Espera pela recepção de uma String.
String texto = in.readUTF();
//Envia o inteiro 6000.
out.writeInt(6000);
//Envia a String “Olá - Socket Servidor.”.
out.writeUTF("Olá - Socket Servidor.");
```

5 – Fechar Streams e socket cliente

```
//Fecha o canal de entrada.
in.close();
//Fecha o canal de saída.
out.close();
//Fecha o Socket que está a atender o cliente.
sock.close();
```

6 – Fechar o socket Servidor

```
//Fecha o servidor.
server.close();
```

Em seguida podemos ver as classes Cliente e Servidor completamente implementadas.

Class Cliente

```
import java.io.*;
import java.net.*;

public class Cliente{
      public Socket client;
      public DataInputStream in;
      public DataOutputStream out;

      public Cliente(){
             try{
                    this.client = new Socket("127.0.0.1",8080);
                    this.in = new DataInputStream(client.getInputStream());
                    this.out = new DataOutputStream(client.getOutputStream());
             }
             catch(IOException e){
                    System.out.println(e.getMessage());
             }
     }

     public static void main(String args[]){
             try{
                    Cliente cli = new Cliente();
                   cli.out.writeInt(3000);
                    cli.out.writeUTF("Olá - Socket Cliente.");
                    int valor = cli.in.readInt();
                    String texto = cli.in.readUTF();
                    System.out.println(valor);
                    System.out.println(texto);
                    cli.in.close();
                    cli.out.close();
                    cli.client.close();
             }
            catch(IOException e){
                    System.out.println(e.getMessage());
             }
     }
}
```

Class Servidor

```
import java.io.*;
import java.net.*;

public class Servidor{
      public ServerSocket server;
      public Socket sock;
      public DataInputStream in;
      public DataOutputStream out;

     public Servidor(){
             try{
                     this.server = new ServerSocket(8080);
                     this.sock = this.server.accept();
                     this.in = new DataInputStream(sock.getInputStream());
                     this.out = new DataOutputStream(sock.getOutputStream());
             }
             catch(IOException e){
                     System.out.println(e.getMessage());
             }
     }

     public static void main(String args[]){
             try{
                     Servidor serv = new Servidor();
                     int valor = serv.in.readInt();
                     String texto = serv.in.readUTF();
                     System.out.println(valor);
                     System.out.println(texto);
                     serv.out.writeInt(6000);
                     serv.out.writeUTF("Olá - Socket Servidor.");
                     serv.in.close();
                     serv.out.close();
                     serv.sock.close();
                     serv.server.close();
             }
             catch(IOException e){
                     System.out.println(e.getMessage());
             }
     }
}
```

Aqui ficam as principais bases desta matéria, a partir daqui é usar a imaginação. Como sugestão para a aplicação dos novos conhecimentos adquiridos, sugiro que experimentem a criação de um simples software de conversação ('chat').

Estruturas de dados

Nesta série de artigos vamos deslindar um assunto que para muitos é difícil de encaixar, enquanto para outros é como somar 1 + 1. No primeiro capítulo desta série de artigos vamos abordar os sistemas numéricos.

Como a maioria dos leitores sabem, o computador não interpreta os números como nós e tem as suas limitações aritméticas. Isto porque o processador apenas trabalha com impulsos eléctricos e estes só podem tomar 2 valores: 0 e 1, representando assim a ausência e a presença, respectivamente, desses ditos impulsos. Portanto, estamos diante um sistema numérico binário.

Naturalmente, como nós pensamos num número sempre no sistema decimal e limitamos a ter uma imagem mental de quantidade, não nos apercebemos que de facto um número, por exemplo o 123, pode ser representado da seguinte forma:

```
1*10^2 + 2*10^1 3*10^0  = 100 + 20 +
3 = 123
```

E os números fraccionários? Simples, vamos ver o exemplo 123,456:

```
1*10^2 + 2*10^1 + 3*10^0 + 4*10^-1 +
5*10^-2 + 6*10^-3
```

Ou seja, cada dígito à esquerda da vírgula representa um valor entre 0 e 9 multiplicado por uma potência de base 10 cujo expoente aumenta com a distância à vírgula. Nos dígitos à direita da vírgula é um valor de 0 a 9 multiplicado por uma potência de base 10 com um expoente negativo que cresce à medida que se afasta da vírgula. Sim, lá vem a Matemática! Mas não se preocupem muito com isso, trata-se de Matemática simples.

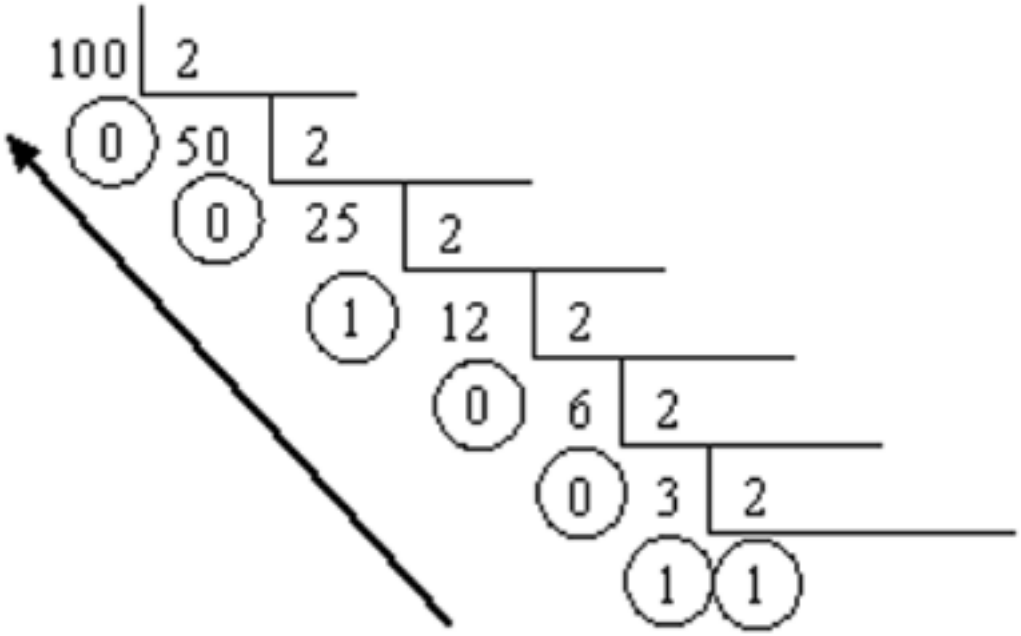
Agora vamos passar ao sistema binário.

No sistema binário só podemos encontrar 2 valores o 0 e o 1, e a partir destes 2 dígitos podemos facilmente obter qualquer valor decimal correspondente. A maneira de converter um binário num decimal é bastante similar ao que fizemos com os números decimais. O sistema binário não permite dígitos diferentes de 0 e 1 e a potência é sempre de base 2 em vez de base 10.
Vamos então converter o seguinte número binário para decimal: 11001010

```
1*2^7 * 1*2^6 + 0*2^5 + 0*2^4 +
1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 =
128 + 64 + 8 + 2 =
202
```

Simples não é?

Para fazer a operação inversa é um pouco pior, temos de dividir sucessivamente um número decimal por 2 até o resultado ser menor que 2. Depois temos de colocar o último resultado e os restos anteriores na ordem como o exemplo a seguir. Ou seja:



O valor binário é 1100100.

As representações dos números binários podem variar. Por exemplo o número 5 no sistema binário representa-se por 101, mas pode ser representado por 0000101 ou 00000101 ou 00000000000000000101, isto porque, tal como no sistema decimal, os zeros a esquerda não têm significado nenhum. Cada digito binário é conhecido por bit (binary digit).

Agora que sabemos converter de binário para decimal e vice versa, vamos ao que interessa.

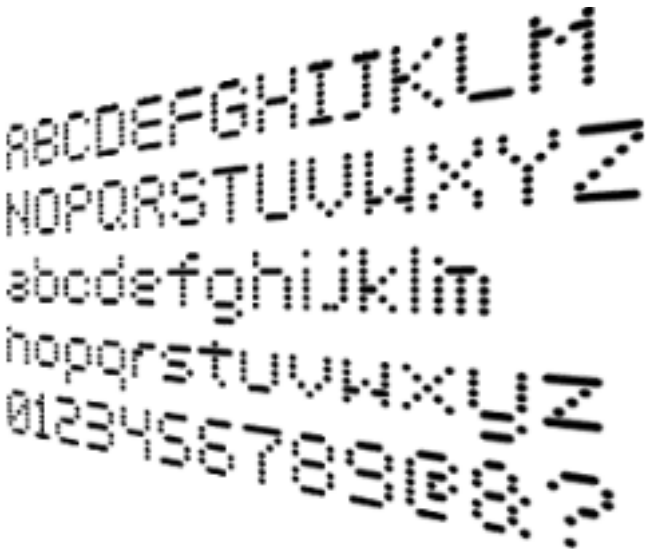
Matematicamente podemos representar qualquer valor decimal em binário sem limitações de tamanho, mas num processador as coisas já não são bem assim, este vai agrupar os bits numa quantidade específica de bits. Ou seja, pode agrupar em 4 bits (conhecido por nibble), 8 bits (conhecido por byte), 16 bits (conhecido como word) e mais adiante.

E vamos ao nosso último exemplo deste artigo:

Representação do número 5 num nibble, num byte e num word.

Nibble: *0101 (4 bits)*
Byte: *00000101 (8 bits)*
Word: *0000000000000101 (16 bits)*

No próximo artigo vamos ver como se organizam estas estruturas que acabámos de descrever.



Diogo Alves

Introdução à linguagem que tem movimentado o mundo open-source

O Python é, como o seu autor diz, uma linguagem de programação interpretada, interactiva, orientada a objectos e dinâmica.

Ao contrário de muitas outras linguagens de programação, a delimitação dos blocos de instruções é feita pelo alinhamento (indentação), não há delimitadores como begin e end como acontece no Pascal, ou { e } da linguagem C. Além disso oferece tipos de dados de alto nível como strings, dicionários, listas, tuplas, classes, entre outros.

A linguagem permite outros paradigmas de programação além da programação orientada a objectos, como a programação funcional. A sintaxe é fácil de compreender e domina-se rapidamente, sendo essas duas características consideradas grandes vantagens da linguagem. Python é, em vários aspectos, semelhante a outras linguagens interpretadas como Perl e Ruby.

Características do Python:

- Sintaxe simples;
- Fácil e rápida aprendizagem;
- Grátis e de código aberto;
- Linguagem de alto nível e interpretada;
- Portabilidade;
- Orientação a objectos;
- Extensível;
- Extensas bibliotecas;
- Excelente suporte e documentação.

Áreas de aplicação:

- Desenvolvimento de aplicações Web;
- Acesso a bases de dados;
- Interfaces gráficas;
- Aplicações científicas e numéricas;
- Educação;
- Programação de redes;
- Desenvolvimento de software;
- Jogos e gráficos 3D.

Antes de tudo, vamos instalar o interpretador da linguagem Python que pode ser obtido na página oficial da linguagem (http://www.python.org). A versão que vamos utilizar ao longo deste artigo é a 2.5 (a última versão estável na altura da escrita do artigo). Se estiver em ambiente Windows basta fazer a transferência do executável de instalação (http://www.python.org/ftp/python/2.5/python-2.5.msi), quem estiver em ambiente Linux ou semelhante tem uma grande probabilidade de já possuir o interpretador instalado, caso não tenham aproveitem agora para o instalar.

Vamos agora ver como correr o primeiro programa em Python, o tradicional Hello World.

Existem dois métodos para o fazer, usando o interpretador interactivo ou um código fonte externo. A vantagem do interpretador é que não têm de gravar e voltar a executar um ficheiro sempre que fazem uma modificação. Mais à frente vamos utilizar ficheiros com o código fonte, pois assim que encerram o interpretador este não grava o que escreveram, sendo por isso útil apenas para perceber como a linguagem funciona e não para escrever programas complexos.

Para iniciar o interpretador pela linha de comandos ou consola basta escrever python seguido de <Enter>. Assim que o fizerem vão ser apresentados com o seguinte (pode ser ligeiramente diferente dependendo do vosso sistema operativo):

```
Python 2.5 (r25:51908, Oct  6 2006, 15:22:41)
[GCC 4.1.2 20060928 (prerelease) (Ubuntu 4.1.1-13
ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

Para fechar o interpretador basta pressionar Ctrl-Z seguido de Enter em Windows, ou Ctrl-D em Linux.

Para aqueles que não se sentem confiantes a trabalhar em linha de comandos, o Python também vem com uma interface gráfica que pode ser usada para programar tanto no modo interactivo como com ficheiros de código fonte. Esta ferramenta chama-se IDLE (Integrated DeveLopment Environment) e é fornecida com a instalação base do Python. Se durante a instalação pediram ao instalador para criar atalhos no menu Iniciar então podem seguir por esse caminho. Caso não tenham instalado atalhos, o IDLE encontra-se em <pasta de instalação do Python>\Lib\idlelib\idle.pyw.

Agora que temos o interpretador iniciado vamos começar a escrever alguns programas. Sempre que o intepretador mostrar >>> significa que está pronto a receber instruções.

Para começar vamos introduzir a instrução print “Hello World”. Como devem ter reparado assim que pressionaram a tecla Enter o Python avaliou o que introduziram e efectuou a devida operação (escreveu Hello World numa linha). Assim que acabou de o efectuar voltou a mostrar os >>>, significando que está pronto a receber novas instruções.

```
>>> print “Hello World”
Hello World
>>> print "Hello", "World"
Hello World
>>> print
<Linha em branco>
```

Como devem ter reparado o comando print serve para escrever algo na linha de comandos. Neste exemplo escrevemos uma string (cadeia de caracteres) mas também podemos escrever listas, tuplas entre outros tipos de dados que vamos conhecer mais à frente...
Se efectuarem o comando print e não fornecerem nada para o interpretador escrever, ele efectua apenas um parágrafo, sendo equivalente a fazerem print de uma string vazia (“”). Também é de notar que se pode fazer print a vários valores numa única instrução, sendo os dados separados por vírgulas.

Tipos de dados

Em Python existe uma enorme variedade de tipos de dados que dão uma enorme flexibilidade ao programador como números, strings (sequência de caracteres), listas (arrays/vectores), tuplas, dicionários (arrays associativos / matrizes), objectos, etc...

Os números subdividem-se em quatro tipos: os inteiros ou decimais (integers), os inteiros grandes (long integers), os números fraccionários (floating point) e os números complexos (complex numbers). A seguir apresentam-se alguns exemplos de operações com números:

```
>>> 5+5
10
```

Neste caso juntamos dois números inteiros sendo o resultado outro inteiro. Também de podem realizar outras operações:

```
>>> 2.5*2
5.0
```

Como viram multiplicamos um número fraccionário por um número inteiro sendo o resultado um número fraccionário. Atenção que para o interpretador 5 não é o mesmo que 5.0.

```
>>> 2**3
8
>>> pow(2, 3)
8
```

Como já devem ter percebido o símbolo ** significa elevar, neste caso elevámos o número 2 ao cubo. Aproveitamos também para apresentar o uso de funções, neste caso a função pow(x, y), sendo x e y os chamados argumentos da função ou seja, os dados que fornecemos à função. Este tema irá ser abordado mais profundamente quando criarmos as nossas próprias funções.

```
>>> 10/3
3
>>> 10.0/3
3.3333333333333335
```

Quando se dividem dois inteiros o resultado é um inteiro, mas quando se misturam inteiros com fraccionários, o resultado é um fraccionário, convém ter este facto em atenção já que é um erro frequente.

Outra função a conhecer muito útil quando estamos a iniciar é a type(), que permite saber o tipo de dados com que estamos a lidar.

```
>>> type(5)
<type 'int'>
>>> type(5.0)
<type 'float'>
>>> type("Python")
<type 'str'>
```

As strings como já foi dito são cadeias de caracteres ou seja podem conter números, letras e símbolos, e o seu conteúdo está sempre limitado por aspas ou plicas:

```
>>> 'Python'
'Python'
>>> "Portugal"
'Portugal'
>>> """Python
... Portugal
... P@P"""
'Python\nPortugal\nP@P'
```

Esta última string construída por três aspas é considerada uma string de várias linhas, reparem que quando iniciamos a string e damos um <Enter> o interpretador passa automaticamente para a linha seguinte mostrando ... em vez das habituais >>>. Outro pormenor interessante é que a string final devolvida pelo interpretador tem apenas uma linha aparecendo o \n (new line) sempre que ocorre uma mudança de linha. Também podemos mudar de linha manualmente sem usar este último tipo de strings:

```
>>> 'Python\nPortugal'
'Python\nPortugal'
>>> 'Triton'Portugal'
  File "<stdin>", line 1
    'Triton'Portugal'
                    ^
SyntaxError: invalid syntax
```

Aqui podem ver outro pormenor interessante sobre as strings que ainda não foi referido. Como fazemos se quisermos usar aspas ou plicas dentro de uma string? Se usarmos assim sem mais nem menos o interpretador confunde-se, não sabendo onde acaba a string, mas se usarmos tipos diferentes de aspas/plicas no conteúdo dos que usamos para limitar a string então já é permitido:

```
>>> "Triton ' Portugal"
"Triton ' Portugal"
>>> 'Triton " Portugal'
'Triton " Portugal'
>>> """Triton '"  Portugal"""
'Triton \'"  Portugal'
```

Neste último caso podemos ver como é que o Python representa a plica numa string. Usamos o chamado caracter de escape, a barra para a esquerda, para representar os tais caracteres não permitidos normalmente ou caracteres especiais (\n).

```
>>> print "Triton \"  Portugal"
Triton "  Portugal
>>> print "Uma barra \\"
Uma barra \
>>> "abc".upper()
'ABC'
```

No primeiro exemplo usamos uma aspa no meio de uma string limitada pelo mesmo tipo de aspa. Para isso fazemos o escape da aspa, usando para isso o \". Para fazemos escape à barra usamos duas barras, como se verifica no segundo exemplo.

As strings apresentam um conjunto de métodos que podemos utilizar, como fizemos no último exemplo, chamando o método .upper() que faz com que a string seja convertida para letras maiúsculas. Para conhecer os métodos de qualquer tipo de dados usa-se a função dir(), por exemplo, dir('Foo') vai mostrar todos os métodos da string (neste caso 'Foo').

Até agora não temos guardado os valores das expressões que introduzimos, para isso usamos as variáveis. Uma variável é apenas um nome que aponta para o endereço de memória onde vamos guardar a informação.

Para guardar um inteiro em memória usamos uma variável, para isso temos de lhe atribuir um nome para que nos possamos referir a esse valor no futuro:

```
>>> numero = 1
```

É tão simples como isto, basta escrever o nome da variável seguido de um igual (=, atribuição) e seguido do que queremos guardar.

```
>>> print numero
1
>>> inteiro = numero + 1
>>> inteiro
2
```

Agora cada vez que nos referirmos à variável número é como nos estivéssemos a referir ao valor que ela guarda em memória, pois cada vez que o interpretador tem de processar a variável acede à memória e substitui pelo devido valor guardado.
As variáveis podem ser usadas para guardar todos os tipos de dados:

```
>>> frac = 2.0*2
>>> print frac
4.0
```

Neste caso antes de guardar o valor o interpretador avalia a expressão (2.0*2) sendo o resultado um número fraccionário.

Graças á simplicidade de Python, esta linguagem é muitas vezes aconselhada para quem entrar no mundo da programação. Também por isso, é possível encontrar muita informação (documentação, artigos, exemplos...) livres na internet, sobre ela. Aqui têm alguns links interessantes:

- http://www.diveintopython.org
- http://www.onlamp.com/python
- http://wiki.python.org/moin
- http://pt.wikipedia.org/wiki/Python

João Matos

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 aos Exploits

Um exploit não é nada mais que um código capaz de explorar uma falha num segmento de código ou software. Do inglês, significa literalmente em português "explorar".

No mundo da segurança informática, denomina-se "exploit" um método capaz de tirar proveito de um bug (falha) de um software provocando comportamentos não pretendidos do software, frequentemente para conseguir escalar privilégios, obter controlo do sistema ou negar serviços (DoS). Geralmente utilizados em milhares de sistemas diariamente, os exploits são a fonte de grande parte dos ataques ocorridos localmente e remotamente nos sistemas existentes. Estes podem ainda tomar formas e poderes bastantes variados. Pode ser um programa executável, uma mensagem num determinado protocolo de rede ou até mesmo uma mensagem escondida num e-mail.

Neste artigo vamo-nos focar no tipo de falha "Buffer Overflow" e utilizaremos a linguagem C para demonstrar.

Como funcionam os exploits?

Os exploits quase sempre fazem proveito de uma falha conhecida como buffer overflow (sobrecarga da memória buffer).

O buffer overflow acontece quando um programa grava dados numa determinada variável passando, porém, uma quantidade maior de dados do que estava previsto pelo programa. Essa situação pode possibilitar a execução de um código arbitrário, necessitando apenas que este seja devidamente posicionado na área de memória do processo.

Abaixo temos um exemplo de um programa vulnerável a um ataque de buffer overflow. O problema está na segunda linha da função ProcessarParam, que não limita o tamanho do argumento recebido (arg).

```
void ProcessarParam(char *arg);

void main(int argc, char *argv[]) {
  if (argc > 1){
    printf("Param: %s\n",argv[1]);
    ProcessarParam(argv[1]);
  }
}

void ProcessarParam(char *arg) {
  char buffer[10];
  strcpy(buffer, arg);
  /* BUG: se a string contida em
  arg tiver mais que 10 carateres
  existirá um Buffer Overflow */
  printf(buffer);
}
```

O buffer overflow, quando ocorre de forma aleatória, normalmente causa um erro fatal/crash na aplicação. No Windows XP esta situação gera uma janela de erro, e no Linux gera a conhecida segmentation fault com core dump (dá-se um core dump quando o sistema consegue guardar o estado do programa antes de surgir a falha, sendo o core o ficheiro guardado). Porém, quando correctamente induzido pelo atacante, o buffer overflow pode permitir que se execute código malicioso que terá os mesmos privilégios de execução da aplicação a ser atacada, que geralmente são privilégios de administrador.

Para entender completamente como o buffer overflow é explorado para se obter acessos indevidos ao sistema, seria necessário compreender como é que os processos são organizados na memória, no qual cada arquitectura de hardware, sistema operativo ou compilador pode organizar de forma diferente.

Buffer overflow é apenas um dos muitos tipos de vulnerabilidades possíveis num software. Sendo alguns deles: heap overflow, integer overflow, return-to-libc attack, format string attack, race condition, code injection, SQL injection, cross-site scripting e cross-site request forgery. Geralmente um exploit apenas toma vantagem de uma única vulnerabilidade de software, o que torna por vezes normal serem utilizados vários exploits em simultâneo: primeiro para ganhar acesso de nível reduzido, depois para escalar privilégios repetidamente até obter privilégios máximos, nomeadamente Administrador/root.

Normalmente um único exploit pode apenas ser utilizado para tomar vantagem numa única vulnerabilidade de software. Habitualmente, quando um exploit é publicado, a vulnerabilidade é corrigida através de uma correcção (patch) e o exploit torna-se obsoleto para novas versões do software. Esta é a razão pelo qual alguns hackers não publicam os seus exploits, mantendo-os privados. Tais exploits são geralmente referenciados como exploits 0day e obter tais exploits é o desejo principal dos atacantes inexperientes, geralmente chamados 'script kiddies'.

Projecto Metasploit?

O Projecto Metasploit é uma plataforma online open-source, disponível para download, que permite controlar e utilizar exploits publicados online.

Escrito em Perl, com componentes em C, Assembler e Python, o Metasploit tornou-se uma ferramenta famosa por facilitar o uso de exploits a todos. Criticado pelos experts da área como uma ferramenta que vem facilitar o trabalho aos script-kiddies e assim aumentar o número de sistemas sob ataques.

Com esta ferramenta, a fase de procura e desenvolvimento dos exploits é praticamente eliminada visto que se torna bastante fácil de descarregar novos exploits para utilizar na plataforma.

Programar em Segurança

Existem muitos tipos de vulnerabilidades, cada uma delas ocorre devido a um erro do programador. Geralmente causadas pela má concepção, falta de conhecimento, entendimento do funcionamento das funções/bibliotecas utilizadas de outros programadores ou até mesmo falhas aritméticas inesperadas. Com isto pode-se facilmente concluir que estas vulnerabilidades apesar de serem bastante conhecidas, não são geralmente entendidas, e isto explicaria o porquê de continuarem a aparecerem enormes quantidades destas falhas nas aplicações de software.

Apesar disto, temos de entender que todos os tipos de vulnerabilidades, nomeadamente as faladas anteriormente são previsíveis. Talvez num tempo próximo no futuro, as condições que permitem estas falhas existirem, venham a ser "corrigidas" e eliminadas, ficam aqui alguns métodos que podem ajudar qualquer programador a prevenir estas:

1. Utilizar diferentes linguagens. Linguagens de programação que fornecem automaticamente verificação de limites como Perl, Python, Java, Ruby, etc. É verdade que estas existem, porém isto por vezes torna-se impossível quando se considera que praticamente todos os sistemas operativos modernos são escritos em C. A mudança de linguagem torna-se particularmente crítica quando é necessário acesso de baixo-nível ao hardware. A boa notícia é que as linguagens estão a evoluir, e a segurança tornou-se um assunto sério. Por exemplo, a Microsoft com a sua iniciativa .NET, rescreveu por inteiro o Visual Basic e Visual C++ com a segurança em mente. Adicionalmente, a linguagem Visual C# que foi desenhada por completo com a segurança em mente.

2. Eliminar o uso de funções de bibliotecas com vulnerabilidades. Linguagens de programação, são tão vulneráveis como o programador permite que sejam. No exemplo dado, no ínicio, utilizámos uma função vulnerável da Standard C Library (strcpy). Esta é uma, de várias, funções existentes na biblioteca que falham em verificar o comprimento/limite dos seus argumentos. Por exemplo, poderíamos ter corrigido a nossa aplicação alterando unicamente uma linha de código:

```
// substituindo:
strcpy(buffer, arg);
// por:
strncpy(buffer, arg, 10);
```

Esta simples alteração, informa o strcpy() que o buffer de destino só tem um tamanho de 10 bytes, e que deve descartar quaisquer dados após este comprimento.

3. Implementar e construir segurança dentro do código. Pode demorar mais tempo, e consome mais esforço, mas o software pode ser construído com a segurança em mente. Se no exemplo anterior, tivéssemos adicionado um passo extra, atingiríamos ainda um melhor nível de segurança:

```
strncpy(buffer, arg, sizeof(buffer));
```

Novamente, isto pode remeter a verdadeira questão, de como os programadores são educados. Será a segurança ensinada, ou encorajada? Será dado o tempo extra necessário para implementar a segurança adequada? Tipicamente, e infelizmente, a resposta é não.

4. Utilizar módulos de bibliotecas seguras. Bibliotecas de segurança em strings estão disponíveis em linguagens como C++. Por exemplo, a C++ Standard Template Library (STL) oferece a classe String. Esta classe oferece funções internas que são seguras no tratamento das strings, e deve ser preferida em relação às funções usuais.

5. Utilizar bibliotecas disponíveis (Middleware). Existem várias bibliotecas de "segurança" disponíveis para utilização. Por exemplo, a Bell Labs desenvolveu a "libsafe" que protege a utilização de funções inseguras. Libsafe funciona na estrutura da stack, e permite assegurar que quando uma função é executada, o endereço de retorno não é alterado. No entanto, como muitas outras bibliotecas, esta não é imune a falhas e deve ser sempre utilizada a última versão.

6. Utilizar ferramentas de análise do código. Foram feitas várias tentativas de criar uma aplicação que executasse uma análise no código fonte e tentasse encontrar potenciais falhas, inclusive Buffer Overflow's. Uma aplicação exemplar chama-se PurifyPlus criada pela Rational's (http://www.rational.com) que executa análises a código escrito em Java, C ou C++ e detecta várias vulnerabilidades.

7. Utilizar ferramentas de optimização do compilador. Praticamente um conceito novo, várias extensões foram recentemente feitas disponíveis para funcionar directamente com o compilador que permitem monitorizar o comportamento do RET (endereço de retorno duma determinada função) e salvaguardar este valor de potenciais alterações. Stack Shield (http://www.angelfire.com/sk/stackshield) e (hhttp://www.research.ibm.com/trl/projects/security/ssp).

8. Actualizar o sistema operativo e aplicação. Talvez a melhor defesa é manter-se ofensivo e informado. Novas vulnerabilidades são descobertas e reportadas todos os dias. Aplicar as devidas alterações e actualizações às aplicações é fundamental. Por exemplo, recentemente foi descoberto uma falha na API – MessageBoxA – para quem desenvolve aplicações em Windows esta função é bastante familiar, pois é utilizada para mostrar mensagens de erro/aviso/informação na plataforma. Este exploit é agora conhecido como o primeiro exploit do Windows Vista (abrange o Windows XP/2003/Vista). Um programador que utilize esta API para criar mensagens em que de alguma forma é permitido ao utilizador fornecer o texto a ser colocado na mensagem, têm uma potencial falha na aplicação que permite um atacante bloquear o Sistema Operativo. Mais informações em http://www.securiteam.com/windowsntfocus/6D00R0AHPK.html.

Finalizando

Educação é a chave no percurso de tornar uma aplicação segura. Grande parte dos programadores sabem da necessidade de verificar dados introduzidos pelo utilizador, verificar os limites nas operações de dados, entre outros, mas poucos têm a noção das consequências da falta destas atenções. É necessário conhecer estas consequências para podermos adoptar as devidas práticas. Por vezes não se trata apenas de uma potencial falha ou crash na aplicação, mas sim dos riscos de segurança que podem causar aos seus utilizadores.

Devem existir métodos e ciclos no desenvolvimento do software, em que o testar do software tem um papel importante. Uma maior atenção deve ser dada a todos os dados obtidos do utilizador, quer seja do teclado, ficheiro, socket, pipe, etc.

Sites aconselhados ao leitor interessado:

- http://en.wikipedia.org/wiki/Exploit_(computer_security)
- http://insecure.org/stf/smashstack.html
- http://julianor.tripod.com/lamagra-bof.txt
- http://www.milw0rm.com/
- http://www.metasploit.com/

Fábio Pedrosa

Clustering

Clustering é o agregar dois ou mais computadores de modo a que estes funcionem em paralelo para um mesmo fim. Deste modo temos o que é chamado cluster, um grupo de máquinas interligadas via (por exemplo) rede local que conseguem comunicar entre si.

Necessidade e Importância

Cada vez mais se utilizam computadores para resolver os mais diversos problemas. À medida que estes vão aparecendo e crescendo em complexidade é necessário ir encontrando novos métodos para ser bem sucedido na sua resolução. Sendo que a abordagem inicial mais comum é a de tentar achar um melhor algoritmo esta nem sempre é conseguida. Muitos problemas que surgem hoje em dia são simplesmente demasiado grandes em complexidade (mesmo com algoritmos bastante trabalhados e optimizados) para serem resolvidos por uma máquina isolada. É neste ponto que entra o clustering. Se conseguimos achar uma solução mas utilizando uma só máquina não a obtemos em tempo útil, então vamos paralelizar o processamento de modo a que este seja distribuído por mais que uma máquina.

Conceitos Básicos

O cluster parte dum princípio tão simples que qualquer pessoa pode ter um em casa. A partir do momento em que ligamos dois computadores e de algum modo conseguimos ter comunicação entre ambos (via rede local, Wireless, USB, Firewire, entre outros) é possível dividir tarefas. Esta divisão de tarefas é o ponto central da ideia por trás de um cluster.

Vamos partir de um exemplo simples: supondo que temos uma aplicação que executa duas somas independentes. Se só tivéssemos um computador o procedimento normal seria a máquina fazer primeiro uma soma e depois a outra, mas vamos supor que temos dois computadores que conseguem comunicar um com o outro. Passamos então ao clustering, um deles (o master) assume a função de distribuir trabalho entre os dois. O outro (slave) apenas recebe “ordens” do que tem a fazer. (Esta divisão de tarefas é muitas vezes crítica e tem de ser muito bem programada pelo programador.) Neste caso a divisão seria simples, o master ficava com uma das somas para executar e daria a outra ao slave, que a recebia pelo meio de comunicação utilizado e devolvia o resultado pelo mesmo canal. No final o master teria ambos os resultados das contas e poderia seguir o normal fluxo de execução. De notar que ambas as somas seriam feitas ao mesmo tempo, mas em máquinas diferentes, demorando portanto metade do tempo inicial (sem contar com as overheads da comunicação).

Este é o princípio básico do clustering e pode-se ver que há diversos factores a ter em conta quando montamos um cluster: desde ligação entre máquinas (latência e largura de banda), número de máquinas interligadas (também chamados nós), capacidade de paralelismo da aplicação até ao custo total de todo o sistema e, muito importante, a função do sistema, pois a configuração da máquina varia muito com a função final que o cluster irá desempenhar.

Modelos de Clustering

Com o passar dos anos o clustering evoluiu desde uma ideia inicial mais focada em resolver grandes problemas para outras áreas, onde o que tem mais importância é a conectividade entre máquinas, segue-se uma lista das áreas onde a importância do clustering (de um ou outro modo) tem crescido.

Computação Paralela

Consiste na divisão por várias máquinas de um mesmo problema. Todas trabalham em conjunto e cada uma resolve uma pequena parte desse mesmo problema. Geralmente para este tipo de trabalho apenas é necessário CPU Time por parte de cada computador, sendo o acesso a discos baixo ou nulo (dependente, no entanto, do problema em questão).

Distribuição de Workload

Aplica-se o clustering quando temos uma máquina a fornecer determinado serviço e, por alguma razão, esta deixa de conseguir (ou nunca conseguiu) dar resposta (satisfazer todos os pedidos). Montam-se então várias máquinas a fornecer esse mesmo serviço e dividem-se os pedidos entre elas. Um exemplo: uma máquina a correr um Web Server (HTTPd) pode não consegui satisfazer todos os pedidos de quem visita as páginas alojadas em disco. Neste caso existe a hipótese de montar uma outra máquina com o mesmo Web Server e uma cópia exacta do disco do servidor inicial. (As máquinas encontram-se interligados e uma delas (slave) irá modificar o seu conteúdo em disco sempre que algo for mudado no master.) Finalmente podemos direccionar os visitantes para um ou outro servidor conforme a afluência de cada computador, evitando a sobrecarga de cada um deles.

Sistemas Tolerantes a Falhas

Nestes sistemas temos geralmente uma máquina inicial que corre as aplicações necessárias, e uma segunda que substitui a primeira no caso desta falhar. Este tipo de esquema tem bastante aplicação em base de dados, onde um primeiro servidor é dado como principal e corre a aplicação normalmente, existindo um segundo que vai mantendo uma cópia da informação e assume o papel principal caso haja uma falha de funcionamento no primeiro. Isto permite manter todo o sistema a funcionar sem prejuízo para utilizadores. Esta é uma configuração simples e existem numerosas formas de dividir os dados, podendo os computadores estar em salas, edifícios, países ou até continentes diferentes, de modo a evitar catástrofes naturais em larga escala ou acidentes físicos locais.

Armazenamento em Larga Escala

Utilizado por motores de busca como o Google este tipo de clustering é (em princípio) semelhante à computação paralela. Em vez de se distribuir código a executar aqui são distribuídos dados a armazenar em disco. Como cada máquina não tem capacidade para se ligar a centenas de discos é necessário o uso de varias máquinas, cada uma com vários discos. Este tipo de clustering alivia o processamento que cada máquina gasta em acessos a disco e acelera os processos de leitura. Apesar destas vantagens a sua principal função continua a ser o armazenamento. Se uma dada base de dados for de tamanho considerável esta é também uma opção a ter em conta, e muitas vezes a única. Atenção que neste sistema não se fala em dados replicados (copias de segurança), podendo também ser algo a ter em conta quando se desenha um cluster desta natureza.

Manutenção e Monitorização

Neste caso temos um número elevado de máquinas que queremos monitorizar.

É possível faze-lo "à mão" mas pode ser demorado. A solução é ligar todas essas máquinas a uma nova que corra aplicações de monitorização, de modo a avisar o responsável quando houver falhas. Dependo da natureza do sistema podemos ainda utilizar estas ligações para actualizações de software e manutenção das máquinas monitorizadas. Um exemplo desta situação pode ser encontrado nas empresas de Web Hosting (que fornecem espaço na Internet para criação de páginas). São empresas que, geralmente, contam com grande número de servidores e que, devido à natureza do software que correm, necessitam de actualizações com alguma frequência. Neste caso a monitorização é também útil para determinar se uma máquina está com problemas, isto é crítico para um negócio que tem de estar a correr 24 horas por dia.

Grid Computing

Conceito e definição bastante semelhante à computação paralela mas com uma diferença em muitos casos. É natural usar o termo Grid Computing para referir uma rede de computação paralela sobre a Internet. É na mesma computação paralela mas com algumas limitações ao nível da largura de banda e latência. Exemplos são o GIMPS, SETI@Home, Folding@Home, etc.

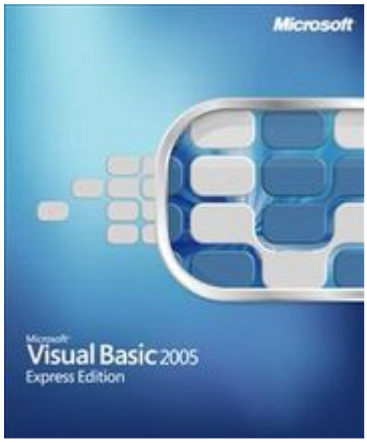
Estes projectos necessitam de grande poder computacional e ao invés de comprarem máquinas que fiquem ao seu serviço optaram por distribuir pelos seus utilizadores pequenos programas que recebem dados de um servidor central, efectuam os cálculos e devolvem os resultados a esse mesmo servidor. Estes são projectos de natureza muito própria, em que uma ligação constante à Internet não é necessária, nem tão pouco que seja rápida ou de baixa latência (algo impossível de garantir). Muitos deles apenas necessitam de uma ligação uma vez por mês de modo a pedir dados e enviar resultados. São um tipo de clustering emergente pois cada vez mais pessoas têm computadores em casa e acesso à Internet. Este modelo tem, como é óbvio, bastantes limitações pelo que a sua maior utilização é junto de projectos menos críticos e onde o dinheiro é uma questão de peso.

Performance do Cluster

A performance de um cluster depende do número (e qualidade) das máquinas que estão interligadas. Quando é necessário mais capacidade de armazenamento ou mais poder computacional são adicionados mais discos ou mais máquinas. O crescimento de performance, a nível computacional, não é linear mas serve os propósitos necessários sem grandes bottlenecks.

O maior problema surge na ligação via rede. Existem duas grandes questões: largura de banda e latência. Os problemas de largura de banda são facilmente resolvidos mudando a ligação ou adicionado linhas extra. O segundo problema, a latência, trás bastantes mais dificuldades. A latência é o tempo que os dados demoram a passar por todo o hardware de um ponto inicial ao ponto final (independentemente do tamanho dos dados, isto é, tempo adicionado ao tempo total de passagem da informação). Este factor é impossível de anular mas pode ser melhorado com hardware de boa qualidade. Esta questão traz problemas com os quais o programador deve contar, tais como a sincronização. É importante ter noção de quanto tempo se gasta em comunicação via rede de modo a minimizar o tempo perdido enquanto os dados são enviados de nó a nó. Quando a programação de cada nó é eficiente a baixa performance da rede é um problema de menor importância. Uma boa técnica para evitar perda de tempo em envios de dados pela rede é a execução de duas instâncias do mesmo programa na mesma máquina. Enquanto uma comunica e espera a recepção ou envio de dados a outra segue o processamento normal.

Pedro Verruma & Luis Rente



Visual Basic .NET 5ª Parte

Como foi referido no artigo anterior, o tema desta edição é relacionado com base de dados, mais concretamente, iremos falar sobre a manipulação de dados entre o VB.NET (http://msdn2.microsoft.com/en-us/vbasic/default.aspx) e o MS SQL Server 2005™ (http://www.microsoft.com/sql).

O ADO.NET é um modelo de acesso de dados (sucessor do ADO – Access Data Object) que nos fornece o acesso a variados "data sources" como por exemplo ao MS SQL Server 2005, que é o que vamos utilizar para os exemplos neste artigo. As classes do ADO.NET estão referenciadas no namespace System.Data mas neste artigo vamos utilizar o namespace System.Data.SqlClient.

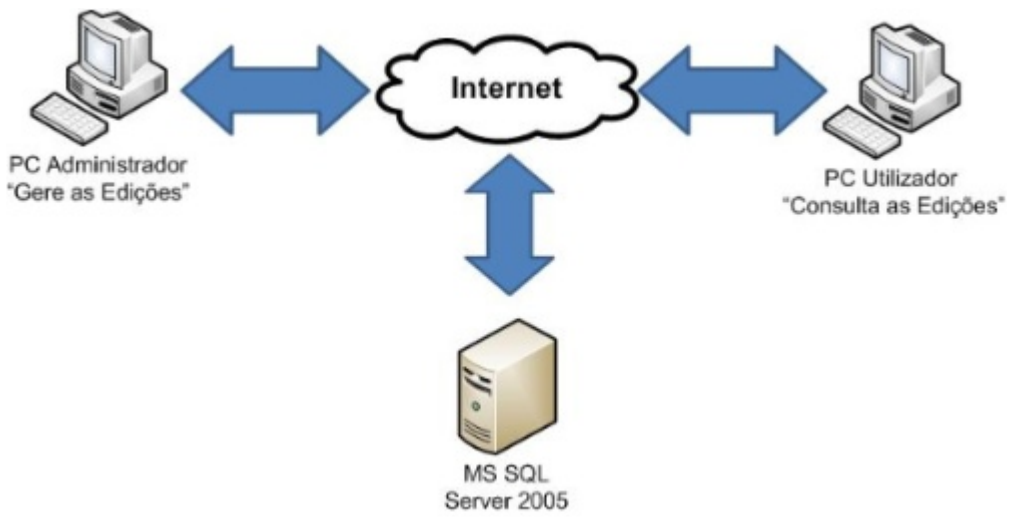
O namespace System.Data.SqlClient dá-nos acesso a todas as classes que correspondem ao provider para o SQL Server, permitindo-nos assim a ligação ao servidor, execução de comandos, leitura dos dados, etc, tudo isto com as suas classes incluídas.

Para exemplificarmos a utilização destas classes, iremos considerar uma situação não real:

"No site da Revista Programar estão listadas todas as edições lançadas até ao momento. Uma aplicação foi desenvolvida para permitir a inserção, alteração e remoção das edições por um administrador que a tenha instalada no seu computador. Todos estes dados estão armazenados numa base de dados em MS SQL Server 2005."

A base de dados utilizada irá conter apenas 2 tabelas. Uma armazena as edições já lançadas e a outra contém todos os comentários inseridos pelos visitantes.

Para podermos comunicar com a base de dados que criamos precisamos de estabelecer uma ligação entre a nossa aplicação e o SGBD . O objecto Connection é usado então nesta situação, permitindo-nos comunicar com a base de dados utilizando classes como a SqlConnection.



```
Dim ligacao As SqlConnection = New SqlConnection("Data Source=XPTO;
        Initial Catalog=RevistaProgramar;User Id=SA;Password=XPTO;")
  Try
    ligacao.Open()
    Console.WriteLine("A ligação teve sucesso")
    Console.ReadLine()
  Catch
    Console.WriteLine("Erro na ligação")
  End Try
```

Na declaração da SqlConnection precisamos também de indicar uma ConnectionString2, usada para estabelecermos a ligação à base de dados em questão.

```
  Data Source=XPTO;Initial Catalog=RevistaProgramar;User Id=SA;Password=XPTO;
```

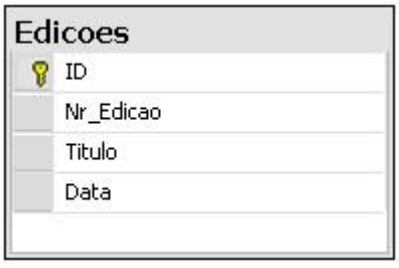
Estabelecida a ligação à base de dados é tempo de começar a manipular os nossos dados. Para isso precisamos de usar a classe SqlCommand para inserir, actualizar ou até mesmo apagar os nossos dados da base de dados. Com esta classe podemos associar uma expressão em T-SQL ou então um Stored Procedure a ser executado na base de dados e uma ligação (SqlConnection)

```
  SqlCommand("Expressão T-SQL ou Stored Procedure", SqlConnection)
```

Para executarmos o SqlCommand podemos usar os métodos ExecuteNonQuery(), ExecuteReader() e ExecuteScalar().

ExecuteNonQuery: Executa a expressão e devolve o número de linhas afectadas na base de dados ExecuteReader: Executa a expressão e constrói um SqlDataReader com os dados devolvidos

ExecuteScalar: Executa a expressão e devolve o valor da primeira coluna na primeira linha dos resultados devolvidos. (Este método é normalmente utilizado quando temos uma expressão que devolve valores agregados como por exemplo "SELECT COUNT(*) FROM tabela".)



Procedimentos:

- Inserir Edição

```
Private Sub InserirEdicao()
  Dim ligacao As SqlConnection
  Dim cmdInserir As SqlCommand
  Dim NrLinhasAfectadas As Integer = 0
  Try
    ligacao = New SqlConnection("Data Source=XPTO;InitialCatalog=RevistaProgramar;
                                 User Id=SA;Password=XPTO;")
    cmdInserir = New SqlCommand("INSERT INTO Edicoes(Nr_Edicao,Titulo,Data)
                                 VALUES(@nr,@titulo,@data)", ligacao)
```

```
    cmdInserir.Parameters.Add("@nr", SqlDbType.Int, 4).Value = Convert.ToInt32(
                                                              txtNumero.Text)
    cmdInserir.Parameters.Add("@titulo", SqlDbType.VarChar, 255).Value =
                                                              txtTitulo.Text
    cmdInserir.Parameters.Add("@data", SqlDbType.DateTime).Value = Convert.
                                                        ToDateTime(txtData.Text)

    ligacao.Open()

    NrLinhasAfectadas = cmdInserir.ExecuteNonQuery()
    If NrLinhasAfectadas > 0 Then
      MessageBox.Show("O registo foi inserido com sucesso", "P@P",
                      MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
      ListarEdicoes()
    Else
      MessageBox.Show("Não foi possível inserir o registo. Tente novamente.",
                      "P@P", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End If

  Catch ex As Exception
    MessageBox.Show(ex.ToString)
  Finally
    ligacao.Close()
  End Try
End Sub
```

- Alterar Edição

```
Private Sub AlterarEdicao()
  Dim ligacao As SqlConnection
  Dim cmdAlterar As SqlCommand
  Dim NrLinhasAfectadas As Integer = 0
    Try
      ligacao = New SqlConnection("Data Source=XPTO;InitialCatalog=RevistaProgramar;
                                   User Id=SA;Password=XPTO;")
      cmdAlterar = New SqlCommand("UPDATE Edicoes " & "SET Nr_Edicao =
         @nr,Titulo = @titulo, Data = @data " & "WHERE ID = " & Convert.ToInt32(
         dgvEdicoes.CurrentRow.Cells("ID").Value), ligacao)

      cmdAlterar.Parameters.Add("@nr", SqlDbType.Int, 4).Value = Convert.ToInt32(
                                                          txtNumero.Text)
      cmdAlterar.Parameters.Add("@titulo", SqlDbType.VarChar, 255).Value =
                                                          txtTitulo.Text
      cmdAlterar.Parameters.Add("@data", SqlDbType.DateTime).Value =
                                                          Convert.ToDateTime(txtData.Text)

      ligacao.Open()

      NrLinhasAfectadas = cmdAlterar.ExecuteNonQuery()
```

```
   If NrLinhasAfectadas > 0 Then
        MessageBox.Show("O registo foi alterado com sucesso", "P@P",
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        ListarEdicoes()
      Else
         MessageBox.Show("Não foi possível alterar o registo.",
                         "P@P", MessageBoxButtons.OK, MessageBoxIcon.Error)
      End If

    Catch ex As Exception
      MessageBox.Show(ex.ToString)
    Finally
       ligacao.Close()
    End Try
End Sub
```

- Remover Edição

```
Private Sub RemoverEdicao()
  Dim ligacao As SqlConnection
  Dim cmdRemover As SqlCommand
  Dim NrLinhasAfectadas As Integer = 0
    Try
      ligacao = New SqlConnection("Data Source=XPTO;Initial Catalog=RevistaProgramar;
                                   User Id=SA;Password=XPTO;")
      cmdRemover = New SqlCommand("DELETE FROM Edicoes " & "SET Nr_Edicao =
            @nr,Titulo = @titulo, Data = @data " & "WHERE ID = " & Convert.ToInt32(
            dgvEdicoes.CurrentRow.Cells("ID").Value), ligacao)

      cmdRemover.Parameters.Add("@nr", SqlDbType.Int, 4).Value =
                                                 Convert.ToInt32(txtNumero.Text)
      cmdRemover.Parameters.Add("@titulo", SqlDbType.VarChar, 255).Value =
                                                 txtTitulo.Text
      cmdRemover.Parameters.Add("@data", SqlDbType.DateTime).Value =
                                                 Convert.ToDateTime(txtData.Text)

      ligacao.Open()

      NrLinhasAfectadas = cmdRemover.ExecuteNonQuery()
      If NrLinhasAfectadas > 0 Then
        MessageBox.Show("O registo foi removido com sucesso", "P@P",
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        ListarEdicoes()
      Else
        MessageBox.Show("Não foi possível remover o registo.", "P@P",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
      End If

    Catch ex As Exception
      MessageBox.Show(ex.ToString)
    Finally
      ligacao.Close()
    End Try
End Sub
```

Como podem ver, esta é uma maneira simples de enviar pedidos à base de dados para executar algo.

Uma técnica muito usada, é o uso de parâmetros nos SqlCommands de forma a simplificar e clarificar o nosso código. Em vez de termos a tarefa de concatenar strings, como por exemplo, "SELECT * FROM tabela WHERE Id=" & variavel_id & " Nome='" & variavel_nome & "'", o que faria com que uma expressão mais extensa se torna-se bastante confusa, podemos associar ao SqlCommand um ou vários parâmetros (SqlParameter).

```
cmdInserir.Parameters.Add("@nr", SqlDbType.Int, 4).Value =
                                                    Convert.ToInt32(txtNumero.Text)
```

de: Procedimento "Inserir Edição"

Um SqlParameter na sua declaração mais simples, pode conter um nome ("@nr"), o tipo de dados (SqlDbType.Int) e um tamanho (4). O nome do parâmetro é depois usado na expressão T-SQL com o valor atribuído na propriedade Value. Nesta situação, trata-se de um parâmetro com a direcção de Input.

Após então executarmos o comando, temos depois que nos certificar que tudo correu como previsto e se o que pretendíamos foi realmente executado. Como referimos em cima, o método ExecuteNonQuery() devolve o número de linhas que foram afectadas na execução de uma expressão. A variável "NrLinhasAfectadas" armazena então essa informação, bastando então ser igualada ao método referido.

```
NrLinhasAfectadas = cmdInserir.ExecuteNonQuery()
If NrLinhasAfectadas > 0 Then
  MessageBox.Show("O registo foi inserido com sucesso", "P@P",
                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
```

de: Procedimento "Inserir Edição"

Para podermos alterar ou remover uma edição, precisamos de indicar à nossa aplicação qual é o número do seu registo de forma a executar ambas as expressões correctamente. Como pudemos verificar no procedimento AlterarEdicao() e RemoverEdicao, temos as seguintes expressões associadas aos comandos:

```
UPDATE Edicoes " & _
"SET Nr_Edicao = @nr,Titulo = @titulo, Data = @data " & _
"WHERE ID = " & Convert.ToInt32(dgvEdicoes.CurrentRow.Cells("ID").Value)

DELETE FROM Edicoes " & _
"SET Nr_Edicao = @nr,Titulo = @titulo, Data = @data " & _
"WHERE ID = " & Convert.ToInt32(dgvEdicoes.CurrentRow.Cells("ID").Value)
```

Este ID é atribuído pelo valor armazenado da coluna com o nome "ID" e na linha seleccionada do componente DataGridView (dgvEdicoes), componente este que nos permitir listar todos os registos da tabela Edições.

Procedimento ListarEdicoes()

```
Private Sub ListarEdicoes()
  Dim ligacao As SqlConnection
  Dim daEdicoes As SqlDataAdapter
  Try
    ligacao = New SqlConnection("Data Source=XPTO;Initial Catalog=RevistaProgramar;
                                 User Id=SA;Password=XPTO;")
    daEdicoes = New SqlDataAdapter("SELECT * " &  "FROM Edicoes", ligacao)
    Dim dsEdicoes As DataSet = New DataSet
    daEdicoes.Fill(dsEdicoes, "Edicoes")
    dgvEdicoes.DataSource = dsEdicoes
  Catch ex As Exception
    MessageBox.Show(ex.ToString)
  Finally
    ligacao.Close()
  End Try
End Sub
```

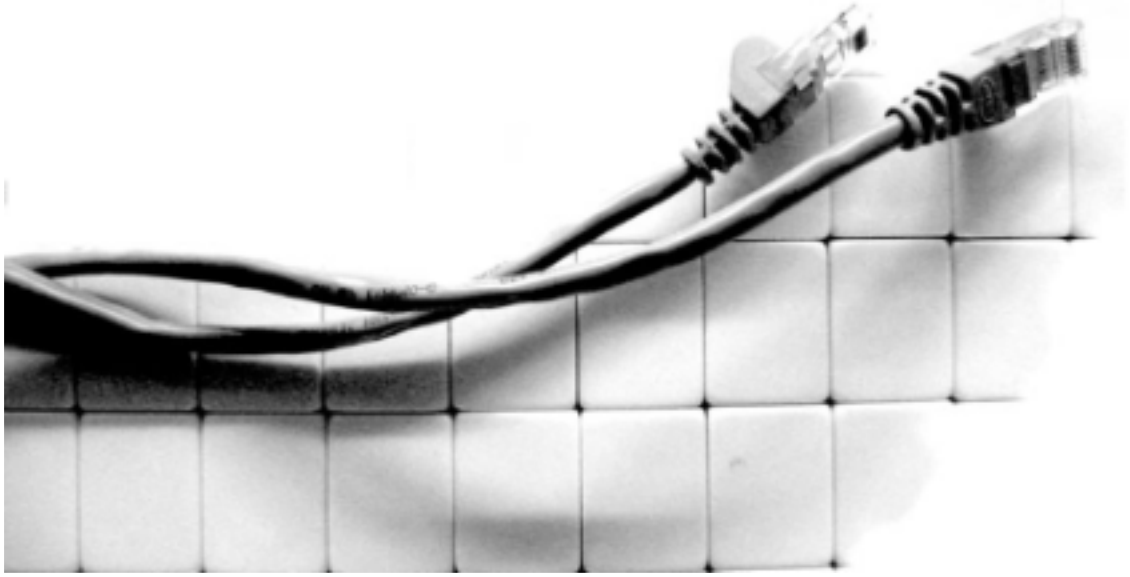
Neste procedimento fazemos uso dos objectos DataSet e DataAdapter. Podemos considerar o DataSet como um conjunto de dados, tal como as estruturas das base de dados, permitindo assim uma maior facilidade na manipulação dos dados, tendo também como vantagens a não necessidade de estar com uma ligação activa à base de dados, e a não interacção directa com a nossa fonte de dados. O DataAdapter por sua vez, é a ponte de ligação entre a nossa base de dados e o DataSet.

Para esta situação, utilizámos o SqlDataAdapter e associámos um comando com uma expressão T-SQL para nos devolver a lista das edições inseridas na base de dados. Após isso, usámos o método Fill, que vai preencher o DataSet com os resultados devolvidos, podendo assim definir o nosso DataSet como o DataSource da DataGridView que vai mostrar as edições.

Feito isto, temos finalmente a nossa aplicação a funcionar correctamente, com a listagem das edições e a possibilidade de inserir, alterar e remover edições.

O projecto de demonstração encontra-se disponível em:

http://www.portugal-a-programar.org/artigos/vb5.rar



Tiago Salgado

Gravação do Desktop

Quantas vezes quis criar um tutorial com a sua distro de GNU/Linux ou a escrever um código que queria partilhar e, em vez de o apresentar com imagens e comentários, pensou em gravar um vídeo consigo a comentá-lo? Criar um projecto para a escola ou universidade em que um vídeo seria mais fácil e prático de apresentar? Criar um vídeo para uma plataforma de eLearning ou bLearning? Ou simplesmente, para mostrar aos amigos os efeitos 3D do seu Desktop?

Neste artigo irá ser apresentada uma solução para tal. O programa usado é o recordMyDesktop.

recordMyDesktop

O recordMyDesktop é um programa criado John Varouhakis, programado em C, lançado em 12 de Julho de 2006 (data da inscrição do projecto no SourceForge) e disponível para download em http://recordmydesktop.sourceforge.net. A licença deste software é GPL (GNU Public License). Está de momento em estado alpha, mais concretamente na versão 0.3.1.

Apesar do programa original ser em modo texto existe uma versão gráfica usando a plataforma GTK.

Anteriormente à escrita deste artigo não existia uma versão portuguesa deste programa. No entanto, os membros desta revista diponibilizaram-se, e, existe agora uma tradução disponível que facilitará a utilização do programa em modo gráfico.

Dependências

- libasound2 (>1.0.10)
- libc6 (>=2.3.2.ds1-21)
- libice6
- libogg0 (>=1.1.2)
- libsm6
- libtheora0
- libvorbis0a (>=1.1.0)
- libvorbisenc2 (>=1.1.0)
- libvorbisfile3 (>=1.1.0)
- libx11-6
- libxdamage1
- libxext6
- libxfixes3
- zlib1g (>=1:1.2.1)

Comandos

A forma genérica de executar o programa é:

```
$ recordmydesktop
```

Isto irá gravar um ficheiro com o nome out.ogg na sua pasta home. Para parar a gravação simplesmente clique Ctrl + C.
Para definir o nome do ficheiro e/ou a localização basta:

```
$ recordmydesktop /home/user/video.ogg
```

O programa também permite definir áreas a ser gravadas. Assim pode usar-se:

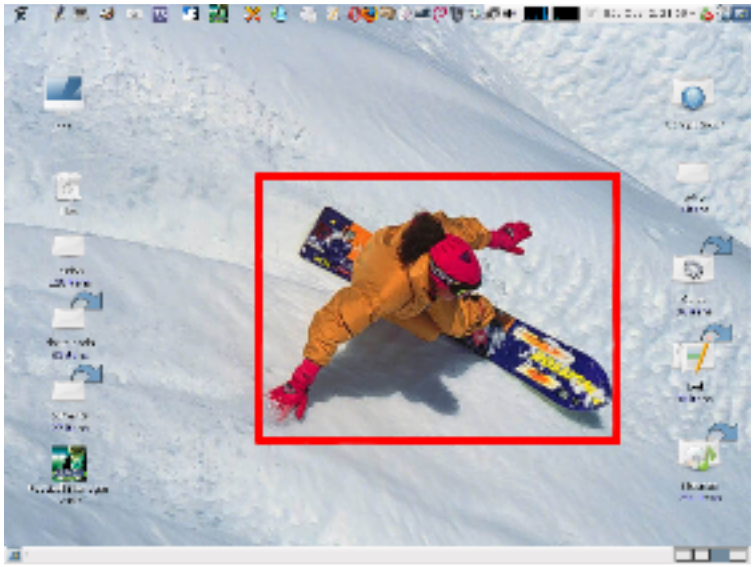
```
$ recordmydesktop -x x-pos -y y-pos
-width largura -height altura -o
video.ogg
```

Tanto a opção x-pos como a opção y-pos referem-se à distância em pixels a partir do canto superior esquerdo.

As opções largura e altura referem-se, como o próprio nome indica, à altura e largura da janela a ser gravada. Se o tamanho da janela inserido for maior que o da resolução actual, será notificado e nada acontecerá. Por exemplo se fosse definido com as seguintes opções:

```
$record recordmydesktop -x 348 -y 235
-width 494 -height 364 -o video.ogg
```

Gravaria apenas esta parte de um Desktop:



Para se ter um cursor diferente pode usar-se:

```
$ recordmydesktop -dummy-cursor cor
```

O valor cor pode ser black (preto) ou white (branco). Irá substituir o seu cursor por um cursor pequeno da cor definida. Ideal para passar despercebido.

Pode ainda ser definido o número de frames por segundo (N) com o seguinte comando:

```
$ recordmydesktop -fps N
```

Para não usar som na sua gravação apenas tem de adicionar a opção --no-sound.

Para o programa codificar o vídeo à medida que vai gravando pode usar a opção --on-the-fly-encoding. Isto irá exigir mais do processador, logo use esta opção apenas se tiver uma boa máquina que aguente as exigências no processamento vídeo.

A qualidade do vídeo é definida pelo seguinte comando em que n é entre 0 e 63(default):

```
$ recordmydesktop -v_bitrate n
```

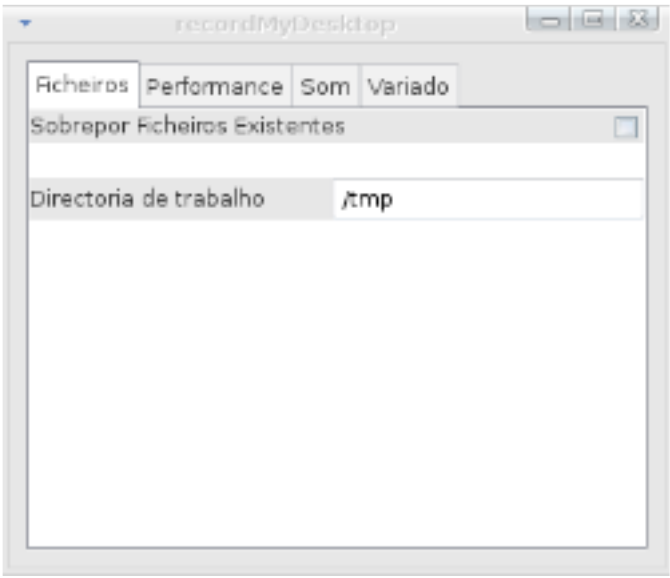
Para definir um atraso a iniciar a gravação basta usar:

```
$ recordmydesktop t[h/m]
```

Como default t assume o valor em segundos (basta pôr um número). No entanto se quiser usar um atraso de h ou minutos basta adicionar a letra correspondente a seguir ao número. Exemplo: 5m, 1h.

Para gravar o Desktop com o compiz ou beryl é necessário também usar a opção --full-shots que tira uma ScreenShot a cada frame.

Modo gráfico



Dependências

- recordmydesktop
- xbase-clients
- python-gtk2
- python-gnome2extras
- python

Com o modo gráfico o programa fica bastante mais prático e fácil de usar. Para seleccionar uma parte da janela basta clicar na imagem e arrastar até onde pretendido. Todas as opções estão também disponíveis no menu "Advanced". No final da gravação, basta clicar no ícone na forma de um quadrado que irá estar colocado na área de notificação do seu Desktop.

Gravar com som

O recordMyDesktop tem a possibilidade de gravar com som. No entanto, nem todos nós gostamos de falar enquanto criamos o tutorial. Por isso, muitas vezes o melhor é fazer o vídeo silenciosamente, e mais tarde adicionar o som. Para tal existem variados programas como por exemplo o Gravador de Som do GNOME e o Audacity.

Legendando o tutorial

Por vezes, por uma questão de acessibilidade a todas as pessoas, o melhor é legendar o tutorial. Existem várias opções, por exemplo o Jubler, o Sabbu e o Subtitle Editor. Vejamos alguns aspectos do Subtitle Editor. O Subtitle Editor é um programa grátis protegido pela licença GPL para editar facilmente legendas.

Pode ser usado para criar novas, para editar, para corrigir ou para sincronizar. Este programa tem um gráfico do espectro de áudio o que facilita em muito o trabalho de sincronizar uma legenda. Com este gráfico do espectro de áudio pode verificar-se as entradas do som, e assim, fazer uma legendagem de forma correcta e fácil sem verificar tantas vezes o filme poupando tempo.

Concluindo o seu vídeo

O formato .ogg, apesar de ser um formato open-source e tão bom como outros, não é muito usado na generalidade. Por isso o melhor é talvez converter o seu vídeo para avi, que é um formato mais conhecido e para além disso torna o vídeo um pouco mais pequeno. Para tal pode usar o mencoder, um pacote do programa mplayer. Basta usar:

```
$ mencoder -idx video.ogg -ovc lavc -oac mp3lame -o video-saida.avi
```

Para adicionar o som pode ser usado o mencoder com o comando -audiofile, como por exemplo:

```
$ mencoder -idx out.ogg -ovc lavc -oac mp3lame -audiofile som.mp3 -o video-saida.avi
```

Basta ter o ficheiro mp3 (ou outro formato de som) que o mencoder faz o trabalho.

No fim de executar todos estes passos terá o seu vídeo pronto a ser partilhado, podendo ser visto, ouvido e lido por todos sem restrições algumas.

Joel Ramos

Desenvolvimento Orientado por Objectos

João Hugo Miranda
José Antonio Almeida

Editora: CentroAtlantico.pt
Colecção: Tecnologias
Páginas: 232
1ª edição: Outubro de 2005
ISBN: 989-615-013-3

Se o leitor não souber o que é desenvolvimento orientado a objectos, domain-driven design ou refactoring e tiver curiosidade em aprender sobre estes temas, então este livro é para si. Se, inclusivé, for um programador que não saiba exactamente como planear um projecto que pretenda começar, então vai encontrar muitas respostas neste livro.

"Desenvolvimento orientado por objectos - Domain-Driven Design, Testes Unitários e Refactoring", de João Hugo Miranda e José António Almeida, editado pelo CentroAtlântico.pt, é um livro que foca temas e conceitos em voga na área da engenharia de software, como o conceito de objectos, domain-driven design, desenvolvimento test-driven, desenvolvimento de interfaces de utilização e refactoring.

Quem quisesse estudar estas temáticas teria de recorrer a livros escritos em inglês, visto que não existiam praticamente referências nenhumas em língua nacional. O livro aborda, portanto, temas considerados de nível intermédio-avançado de forma simples, pragmática e estimulante até para os iniciantes, alargando assim o público alvo da obra. Segue um estilo reflexivo e por vezes também didáctico.

Para além de exporem factos e abordagens, os autores contam histórias, apresentam exemplos práticos, reflectem e discutem os factos. A obra relaciona ainda o paradigma dos objectos com as práticas de programação mais recentes e demonstra que a análise, desenho e a codificação de um sistema de informação não devem ser actividades separadas, como é muito usual acontecer. O livro faz com que o leitor se sinta interessado com estes temas, já que apresenta exemplos do dia-a-dia de um programador, excertos de código devidamente comentados, diagramas e esquemas feitos por profissionais. Os exemplos são em C#, mas qualquer pessoa com conhecimentos mínimos de programação deverá segui-los sem dificuldade.

É recomendado a estudantes de programação e programadores profissionais, obviamente. Pode ainda ser recomendado a gestores de projectos, analistas ou responsáveis técnicos já que parte do texto, especialmente os primeiros capítulos, estabelece o enquadramento conceptual das temáticas debatidas ao longo dos restantes capítulos.

Rui Maia

Google apresenta os 10 termos mais procurados de 2006

O mais famoso motor de busca da internet continua a tradição anual de publicar os dez termos mais pesquisados no seu site.
As duas palavras mais pesquisadas foram Bebo e MySpace, dois portais de comunidades a criação de páginas pessoais e afins, seguidas pelos fanáticos do futebol com "World Cup". O vídeo online também marca presença nas pesquisas dos internautas, no 4º lugar com o site Metacafe e em 7º com a palavra Video.

Para mais informações:
http://www.google.com/intl/en/press/zeitgeist2006.html

1. Bebo
2. MySpace
3. World Cup
4. Metacafe
5. Radioblog
6. Wikipedia
7. Video
8. Rebelde
9. Mininova
10. Wiki

Registo de marcas online

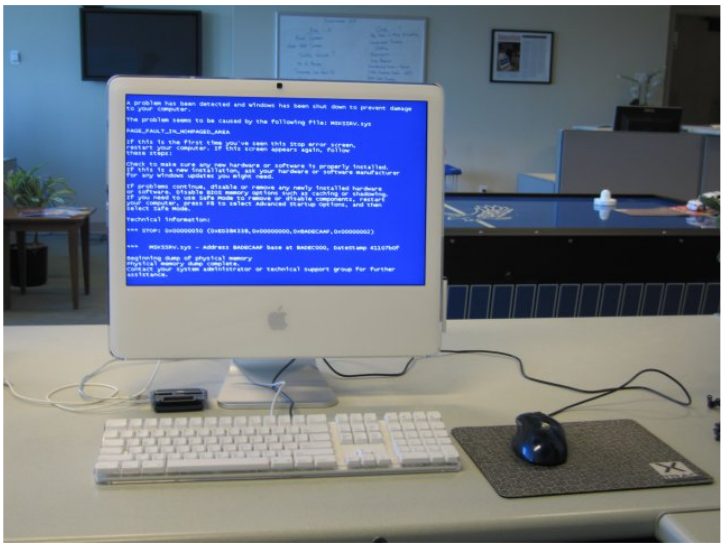
Foi disponibilizado dia 22 de Dezembro do presente ano uma nova área no portal do Instituto Nacional da Propriedade Industrial (INPI), entidade responsável pelo registo de marcas, patentes e afins, que permite o registo Marcas, Designs e Invenções. Este sistema passa a ser mais cómodo para o utilizador, visto não precisar de sair de casa nem de tratar de papelada e simultaneamente mais barato.

http://www.inpi.pt

The CSS Power

Há algum tempo atrás, foi apresentado no fórum do Portugal-a-Programar um site bastante interessante para todos os programadores web que querem aprender CSS e/ou trabalham com ele regularmente, falo do site Maujor (http://www.maujor.com/). Espantoso. Isto é o pensamento que muitos poderão ter ao abrir este site, e explico o porquê: grande quantidade de informação, fiável e, principalmente, em Português. É certo que é português do Brasil, mas ainda há esperança que se desenvolva uma plataforma assim em Portugal. Ficamos à espera...

Dificuldades em largar
o computador nas férias...



Nova ameaça aos Macs

Novo gadget para
a família iPod

Teclado l33t

Estamos a iniciar um novo ano e esta costuma ser uma época propícia às reflexões e à introspecção: olhamos para o que fizemos anteriormente e analisamos de acordo com o que pretendemos fazer no futuro. É uma altura do ano em que, pelo facto de estarmos no início, nos sentimos renovados, prontos a enfrentar tudo outra vez se realmente tiver de ser. Depois das festas de Natal e do Ano Novo, Janeiro chega em força e refazem-se os planos, renovam-se os objectivos. Isto acontece principalmente com as (grandes) comunidades.

Ora, numa conversa com um membro do staff do P@P em finais de Dezembro, nos dias após o Natal e antes de chegar o Ano Novo, reflecti sobre um assunto em que já tinha pensado anteriormente, mas não do ponto de vista pessoal como aconteceu nesta discussão. O meu colega perguntou-me que objectivos, que metas pretendia eu atingir enquanto administrador do P@P. De facto, ponderei bastante na questão e achei que podia responder exactamente com o que tinha colocado nos tópicos que apresentei ao staff, mas quando ia começar a escrever notei que a pergunta se dirigia mais ao que eu achava e menos ao que a comunidade realmente necessitava.

Comecei então por responder que quero fazer com que os programadores do P@P passem o máximo tempo possível a trabalhar no fórum, isto é, pretendo criar condições para que daqui comecem a surgir trabalhos sérios e bem organizados. O meu colega comentou que era uma tarefa difícil, que ia envolver mais gente a trabalhar de perto comigo ou então mais trabalho para os actuais membros do staff. Eu não pude discordar, pelo que continuei a sonhar alto: disse que em 2007 gostava que surgisse o primeiro concurso de programação do P@P, patrocinado com prémios de uma empresa a designar (uma das empresas das parcerias) e que gostava de criar um evento no Verão do género Google Summer of Code, mas exclusivo para membros do P@P.

O meu colega afirmou que estes dois "sonhos" só poderiam avançar se realmente conseguisse realizar o primeiro, o sonho de criar condições para por os programadores do P@P a trabalhar em plataformas da comunidade. Mais uma vez, não pude discordar.

Finalmente, disse que outro grande objectivo meu enquanto administrador da comunidade era fazer com que os projectos agora em arranque ficassem a trabalhar a todo o gás até ao fim do primeiro semestre de 2007. Seria um avanço fabuloso e daria ânimo e muita vontade ao staff e mais possibilidades aos utilizadores que quisessem participar. O colega com quem mantinha a conversa perguntou então se não ia falar do crescimento de utilizadores do fórum, se não ia mais uma vez apontar dados estatísticos. Eu respondi que, no caso de conseguir realizar estes meus sonhos enquanto administrador da comunidade, todas as estatísticas "falariam" de sucesso e de prosperidade e então podíamos comparar com o ano 2006 e prever 2008.

Acontece que estes meus desejos não são de modo algum a prioridade do staff do P@P. Aponto, mais uma vez, a diferença entre objectivos da comunidade de objectivos pessoais enquanto administrador da comunidade. No entanto, sonhar é o primeiro passo para a satisfação pessoal e espero que não seja o último, no nosso caso. Espero que em 2007 possamos todos sonhar uns com os outros, mas mais do que isso, trabalhar no sentido de realizar os nossos desejos. Espero ainda que daqui a sensivelmente um ano, tenhamos todos um grande sorriso na cara e estejamos prontos a encarar outros voos, outros sentidos sempre na direcção da realização pessoal e comunitária. Até lá, espero que tenham um 2007 cheio de bons momentos e que traga prosperidade a 2008. Um grande abraço, em nome do staff do P@P.

Rui Maia

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 informação 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

A Revista PROGRAMAR é um projecto da comunidade Portugal-a-Programar

www.portugal-a-programar.org

www.revista-programar.info