Edição Especial com 2 CDs!

# Aprenda a Programar

www.digerati.com
Ano 1 – Número 1 – R$ 11,90

## + de 80 Tutoriais

**Programe em diversas linguagens**

Apostilas completas de Assembler, C, C++, PHP, Java, SQL, ASP, XML, bancos de dados Oracle e Microsoft, Perl, Python, Linux, algoritmos e redes Novell e Netware

## + de 1.000 Códigos-fonte

**Aprenda a programar analisando scripts prontos**

Inclui códigos do AbiWord, KICQ, Linux Kernel 2.5 e Miranda IM

**Tudo para você aprender a programar**

## C/C++

- Tutoriais completos na revista e no CD
- 120 Códigos-Fonte para análise
- Compiladores

## Dicas de Delphi

Uma coleção com 40 dicas para você aprimorar seu código e

**+ de 100 Códigos-fonte**

e disassemblers, para você estudar o seu programa favorito

**Tudo sobre**

## Cracking

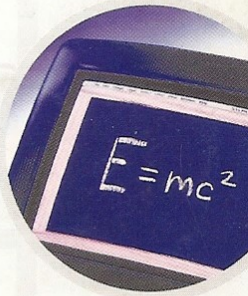
Como são feitos os programas que quebram a proteção de softwares

## Disassemblers

Converta arquivos EXE para códigos-fonte e estude seu programa favorito

Veja mais no verso

ISSN 1678-7315



DIGERATI tech

DIGERATI tech

# Aprenda a Programar

## Edição especial com 2 CD´s

# 1.000 Códigos-fonte para você analisar

## Destaques:

### Top Codes

**7-Zip** Programa de compactação de arquivos compatível com ZIP, RAR, etc.
**AbiWord** Código-fonte do processador de textos open source
**Active Spam Killer** Código-fonte do programa que barra spams do seu e-mail
**DVDrip** Saiba como funciona o programa que quebra a segurança dos DVDs
**KICQ** Código-fonte do comunicador instantâneo para Linux
**Linux Kernel 2.5** O código por trás do Linux
**Miranda IM** Source code do programa de mensagens instantâneas
**The Open AntiVirus Project** Ajude a desenvolver o código do projeto Open AntiVirus
**ZSNES** Código-fonte do emulador de Super Nintendo para Windows e Linux

### Códigos por Linguagem

**C/C++** Mais de 120 programas em forma de código-fonte feitos em C/C++
**Delphi** Mais de 100 programas desenvolvidos em Delphi e seus códigos-fonte
**Pascal** Coleção com mais de 20 códigos-fonte de programas feitos em Pascal
**Visual Basic** Mais de 150 programas com códigos-fonte em VB
**PHP Scripts** Scripts em PHP para análise e estudo
**JavaScripts** Mais de 160 scripts em Java, do nível mais básico ao avançado
**ASP Sources** Mais de 100 scripts e códigos-fonte em ASP
**Perl** Mais de 100 códigos e scripts em linguagem Perl

### TUTORIAIS

*Mais de 80 Cursos e Apostilas com tudo sobre programação. Confira os destaques:*

**Iniciando em C** As bases sobre a principal linguagem para programar aplicativos
**Dicas de Delphi** Toques para você melhorar ainda mais o seu código
**Tudo sobre cracking** Como é quebrada a proteção de programas
**RBT** Curso de Assembly totalmente em português
**Adam's Assembler** Tutorial de Assembler em inglês
**Intro to Assembler** Uma breve introdução à linguagem (em inglês)
**ASM in 80x86 processor** Material sobre Assembler focado para processadores 80x86
**Programação GNU/Linux** Pequeno guia de orientação a programadores
**The Art of Assembly** Tutorial sobre programação em Assembler
**The PC Assembler Tutor** Tutorial ensinando os usos da linguagem Assembler nos PCs
**A HyperGuide to C** Guia rápido sobre a linguagem C
**Programação Shell** Tutorial ilustrado sobre programação Shell
**Apostila C em Word** Apostila sobre a linguagem C em formato de texto do Word
**Notas sobre C** Notas de aulas sobre programação em C
**C Programming Education Package** Guia rápido sobre a linguagem C
**C++ for C Users** Guia de C++ para programadores de C
**Turbo C** Curso básico de programação em Turbo C
**Estrutura em C** Estrutura básica de um programa feito em C
**Virtual School** Curso de programação C da Virtual School
**Transição do Pascal para C++** Curso para programadores que mudaram do Pascal para C++
**PHP na Web** Curso de aplicações Web em PHP
**Introdução PHP** Cursos em HTML com material introdutório à linguagem PHP
**Programando para a Web** Programando para Web com PHP/MySQL
**Java Servlets** Construa sites com Java Servlets Technology
**Java - Orientação a objetos** Tutorial em português sobre Java
**Manual Java** Um manual supercompleto sobre Java
**Programação de Algoritmos** Apostila sobre algoritmos
**Banco de Dados SQL** Tutorial supercompleto feito pela Universidade Federal do Paraná
**Introdução ao SQL** Apostila com os principais comandos SQL
**Portal de Notícias (SQL)** Aprenda a criar um portal de notícias baseado em PHP/MySQL
**SQL Oracle** Apostila de SQL em Oracle.doc
**SQL Server 6.5** Apostila muito boa do SQL Server 6.5
**ASP** Tutorial completo sobre ASP
**Conheça o XML** Saiba mais sobre o XML
**Curso de linguagem PHP** Conheça e aprenda mais sobre a linguagem PHP
**GNU Privacy Guard** Texto para quem se interessa por criptografia
**Learn Turbo C** Programa que o ajuda a aprender Turbo C
**Perl** Texto com informações sobre Perl
**Perl: o início** Tutorial de Perl para iniciantes
**PHP3** Aprenda sobre esta linguagem de desenvolvimento para a Web
**The Zope Book** E-book ensinando tudo sobre o Zope
**Tutorial Python** Tutorial sobre a linguagem Python
**Comandos Básicos do Linux** Manual com comandos básicos do Linux
**Criando pacotes .deb** Manual para implementação de pacotes no Debian
**Dicas para Linux** Várias dicas para utilizar o Linux
**How To de como instalar o Zip** Aprenda a instalar o Zip Drive no Linux
**Integração Windows/Linux com Samba** Saiba como configurar o Samba no Linux para interagir com o Windows
**Linux em redes Netware** Guia detalhado com informações sobre o Conectiva Linux em uma rede Netware
**Curso de Unix** Curso sobre Unix em português
**Novell 4.1** Texto de referência sobre redes Novell
**Samba** Apostila sobre Samba
**Unix avançado e programação C_Shell** Curso avançado de Unix e programação C Shell

### SISTEMAS EMBUTIDOS

**Red Hat eCos** Sistema operacional com código aberto para sistemas embutidos
**Ash** Bourne Shell pequena e simples para Embedded Systems
**ASM Utils** Ferramenta similar ao BusyBox, com vários utilitários
**e3c** Pequeno editor de textos para sistemas embutidos
**iProute** Miniaplicativo para manutenção, configuração e uso de redes
**The MicroWindows Project** Minissistema de navegação com interface gráfica em desenvolvimento
**Sash** Stand-alone shell com comandos de construção
**THTTPD 2.20c** Web server de tamanho super-reduzido para sistemas embutidos
**GNU Zebra** Gerencia clientes de protocolos de rede tipo TCP/IP

### DISASSEMBLERS

*Programas para desmontar e analisar códigos a partir de executáveis*

### GTK

*Pacote com bibliotecas, interfaces visuais e editores para GTK*

### COMPILADORES e UTILITÁRIOS

*Os melhores compiladores para desenvolver programas em C, C++, PHP, ASP, Python, Java, Pascal, XML e até BASIC e COBOL!*

### FERRAMENTAS LINUX

*Mais de 20 programas para você ter tudo do que precisa na programação em Linux*

**Equipamento mínimo:** Processador Pentium II ou equivalente. 64 MB de memória RAM; 16 MB de vídeo, em resolução 800 x 600 pixels; placa de som.

**Informações complementares:** O CD-ROM brinde é composto por uma coletânea de software, freeware, shareware e demonstração. Os requisitos de sistema podem variar de programa para programa. Alguns aplicativos, por motivos alheios à nossa vontade, podem não rodar no Windows XP.

# Crime é não Aprender

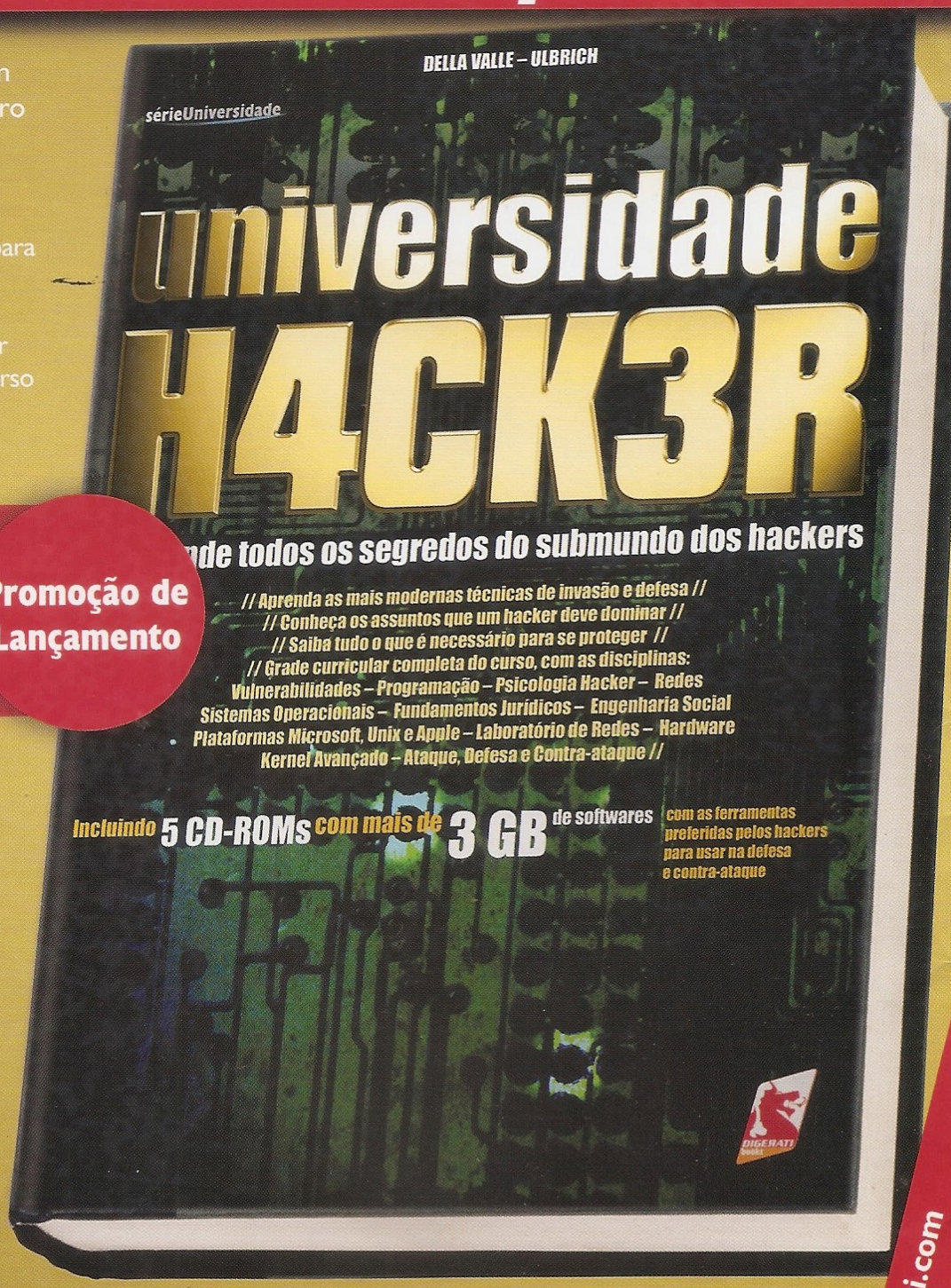
Reunimos dois especialistas em segurança digital para criar o livro mais aguardado do ano:
**Universidade Hacker**

- Aprenda tudo que é necessário para se proteger e contra-atacar
- Conheça os assuntos que um hacker profissional deve dominar
- Grade curricular completa do curso

**Lançamento Nacional**

Fazendo a sua reserva pelo site da Digerati, você pode adquirir qualquer revista da Loja Virtual inteiramente grátis!

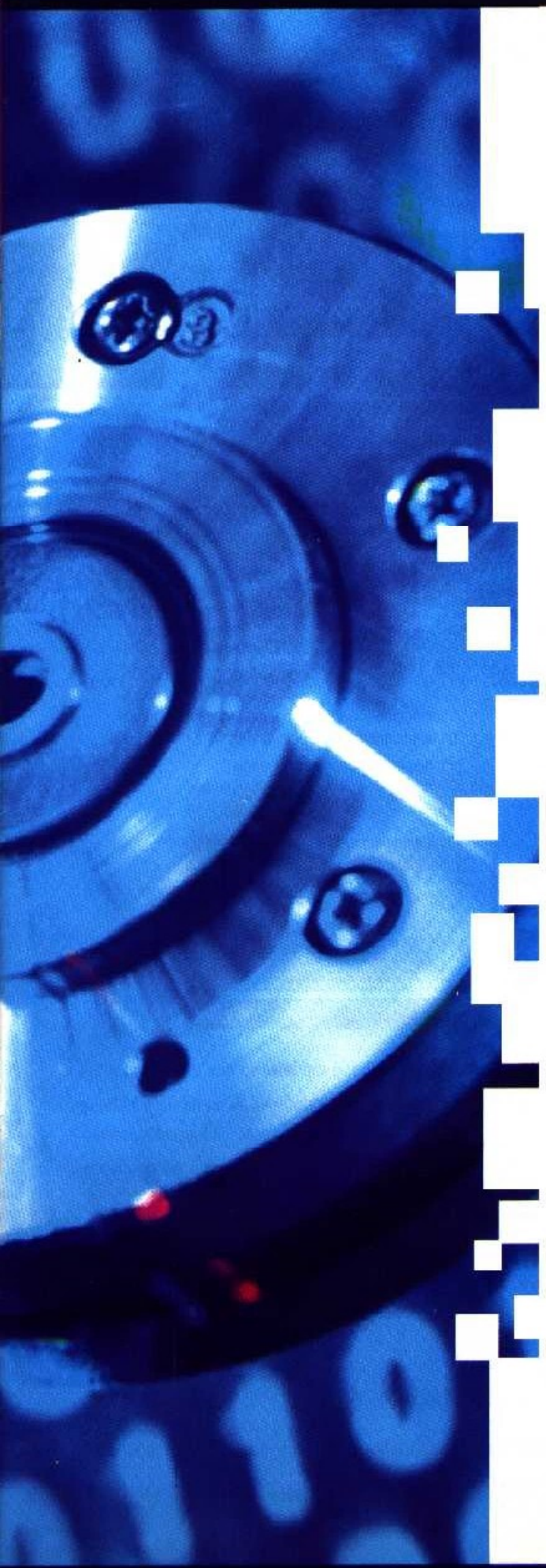
**Promoção de Lançamento**




Livro **Universidade Hacker**
300 páginas por R$ 49,90
nas livrarias ou no site www.digerati.com

DIGERATI

www.digerati.com



O melhor da programação reunida numa só revista. Esse é o objetivo desta revista especial **Aprenda a Programar**. Há muito queremos juntar os melhores programas e tutoriais sobre as principais linguagens de programação. Depois do sucesso das outras revistas da nossa editora voltadas para o tema, nós resolvemos juntar os principais CDs com esses programas.

E foi o que fizemos com esta edição especial. Na revista, você encontrará os principais artigos explicando conceitos sobre essas linguagens, ensinando o básico necessário para iniciar o aprendizado no maravilhoso mundo da programação.

Leia os artigos e explore os dois CDs que acompanham esta edição. Com certeza, você não será o mesmo ao terminar de ler esta revista. Depois, é só escolher a linguagem que mais lhe agrada ou que mais retorno profissional irá lhe dar, e continuar a estudar.

Afinal, o que faz um bom programador é o estudo constante, sem nunca parar. Atualmente, com o desenvolvimento constante de novas técnicas, quem fica para trás não consegue se recuperar.

Bons estudos.

**O Editor**

# Índice

# TUTORIAL BÁSICO DE CRACKING

por Melovis
cr4ck1n6@yahoo.com.br

Primeiramente, algumas recomendações...

Bem, amigos, este tutorial de cracking foi feito apenas com fins explicativos e educativos! Se você quebrar um programa com essas informações, e vendê-lo ou passar para alguém, saiba que isto é um crime de quebra de copyright! O intuito aqui é só ensinar como é fácil e possível quebrar um programa que exija n/s (nome e serial).

## Breve comentário...

Antes de crackear, saiba que *"cada programa é um programa"*, ou seja, cada programa tem uma proteção diferente, uns são bem difíceis, enquanto outros são ridículos.

## Conhecimentos básicos...

Olha, o interessante é saber um mínimo de assembler, mas se você não tem conhecimento nessa linguagem, não se preocupe, pois conseguirá quebrar este exemplo do mesmo jeito!!!

## Ferramentas...

Aqui nós usaremos apenas duas ferramentas:

W32dasm (um dessassembler muito chique!!!)

HIEW (um editor hexadecimal lindo!!!)

## Programa-alvo...

O nosso programa de hoje será nada mais nada menos do que o TÃO famoso mIRC 6.01 (disponível em *www.superdownloads .com.br*)

## Parte I - Testando o registro

Bem, como todo programa shareware, ele possui alguns limites. Usando as informações do superdownloads, para registrar este programa são necessários US$ 20,00 (É DOIDO), mas como todo cracker não paga por programas, nós aprenderemos a registrá-lo pagando nada!!

Publisher: Khaled Mardam-Bey

Publicado em: 24.09.1999

Atualizado em: 13.02.2002

Tamanho: 1,12 MB

Categoria: Chat em Texto

Distribuição: Shareware

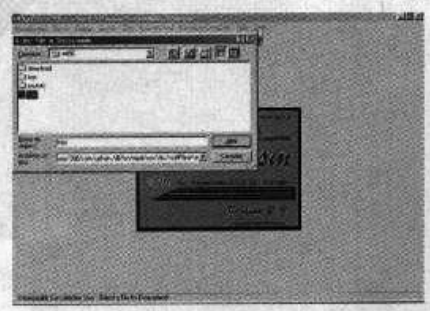
Preço: (US$ 20,00)

Expira em: (não disponível)

Downloads: 202.349

Depois de aberto, o programa aparece assim:

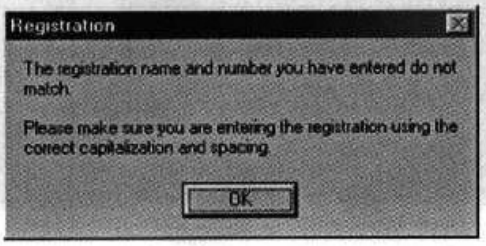
Repare na seta em vermelho, é um letreiro que diz *"Cópia não-licenciada"*. Pois bem, façamos o seguinte: vá ao menu *Help* e clique em *Register*. Aparecerá uma caixa assim:



Na parte *Full Name*, coloque qualquer coisa; na parte *Registration Code* também.

Agora clique em *Register*!

Vai aparecer uma mensagem de erro assim:




Aqui que está nosso ponto de partida! Repare nas linhas sublinhadas de vermelho. Esta frase nos ajudará para quebrarmos o programa. Todo processo de cracking começa pela mensagem de erro que aparece ao usuário quando se erra o registro!

Bem, agora anote a frase sublinhada, feche o mIRC e vamos para a segunda parte.

## Part II - ANALISANDO A ALMA DO PROGRAMA...

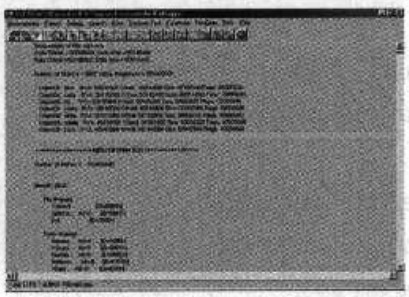
Vamos agora abrir o *w32dasm* e, no menu *Dissassembler*, iremos a *open/select file to dissassembler*. Então, escolheremos o executável do mIRC.

Assim:

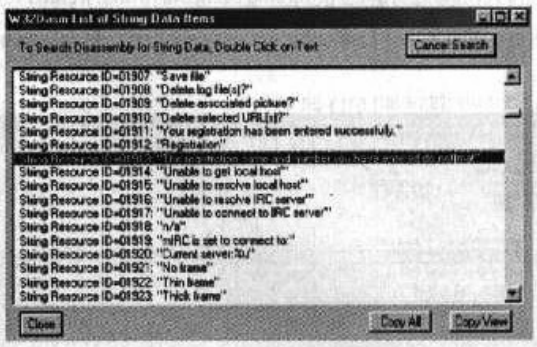

No meu caso, o mIRC está em *c:\program files\mirc\mirc.exe*

Espere carregar e pronto! Em alguns casos, aparece um monte de símbolos estranhos. Mas não se preocupe: é só ir a *dissambler*, *font* e escolher uma fonte legal, tipo *arial* ou *times new roman*.

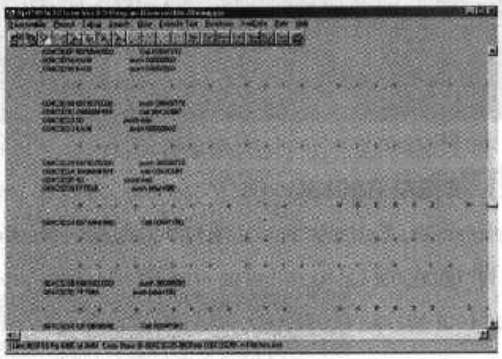
Pronto. Se você fez certo, deverá aparecer assim:



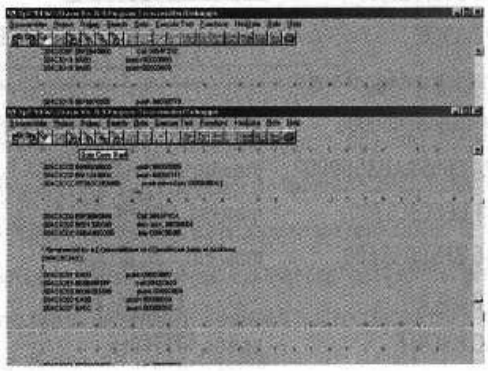
Bem, aqui é a alma do programa. Todo o código está em assembler. Não se preocupe com este monte de instrução agora. Com este programa, nós procuraremos a rotina de registro. E acharemos suas funções para serem analisadas. Lembra-se da frase de erro? Pois é, ela será usada agora. Clique no menu *Refs* e escolha a opção *String Data References*. Nesta janela, estão todas as informações que o programa mostra para o conhecimento do usuário. São as referências de comandos, de erros, etc., ou seja, todas as frases que darão conhecimento para o usuário estão nesta janela, inclusive aquela mensagem de erro...



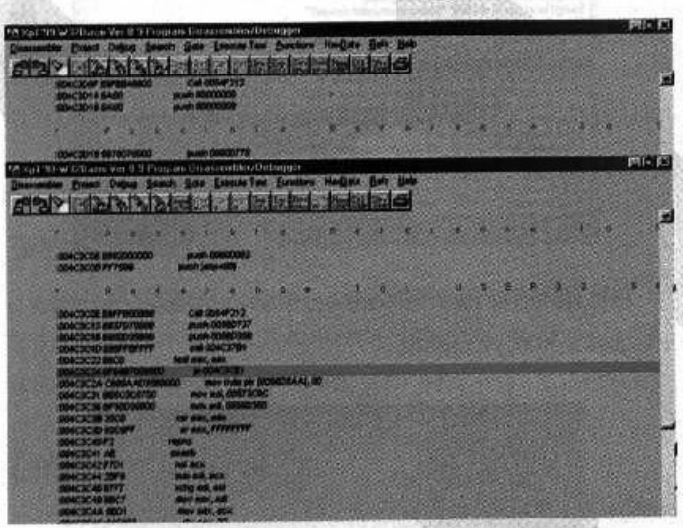

Bem, depois de ter achado a frase, clique duas vezes em cima dela e vá para a rotina de registro.



Aqui é a rotina que nós caímos quando o registro não é aceito. Se subirmos um pouco, perceberemos que essa rotina é executada devido a um call (chamada do código).



Agora veremos porque o programa cai nesta rotina. Iremos ver agora o endereço 004C3C24! É só subir um pouco a barra de rolagem...



## Aqui!!! Vejam agora:

Em assembler, JE significa pular se for igual, ou seja, se o código digitado for igual, ele cai na rotina de erro. Se não for, o programa é registrado com qualquer código!!!

```
:004C3C0E E8FFB50800        call 0054F212
:004C3C13 6837D75600            push 0056D737
:004C3C18 6850D35600            push 0056D350
:004C3C1D E8BFFBFFFF        call 004C37B1
:004C3C22 85C0                  test eax, eax ——à Testa se o
valor do registro e igual a ele
:004C3C24 0F84B7000000      je 004C3CE1à se for, cai na janela de erro.
:004C3C2A C605AAD3560000    mov byte ptr [0056D3AA], 00
:004C3C31 BE6C3C5700        mov esi, 00573C6C
:004C3C36 BF50D35600        mov edi, 0056D350
:004C3C3B 33C0                  xor eax, eax
:004C3C3D 83C9FF            or ecx, FFFFFFFF
:004C3C40 F2                repnz
:004C3C41 AE                scasb
:004C3C42 F7D1                  not ecx
:004C3C44 2BF9                  sub edi, ecx
:004C3C46 87F7                  xchg edi, esi
:004C3C48 8BC7                  mov eax, edi
:004C3C4A 8BD1                  mov edx, ecx
:004C3C4C C1E902            shr ecx, 02
:004C3C4F F3                repz
:004C3C50 A5                movsd
:004C3C51 8BCA                  mov ecx, edx
:004C3C53 83E103            and ecx, 00000003
:004C3C56 F3                repz
:004C3C57 A4                movsb
:004C3C58 6837D75600            push 0056D737
:004C3C5D 6850D35600            push 0056D350
:004C3C62 E8BCFEFFFF        call 004C3B23
:004C3C67 6A00                  push 00000000
```

O que nós faremos agora? Bem, é só substituir o JE por um JNE (pular, se não for igual).

Em hexadecimal, o código de JE é 0F84 e o de JNE é 0F85. E como eu faço isso? Bem, aí é que entra o nosso querido **hiew**. Mas primeiro anote o que aparece na barra de status do w32dasm:

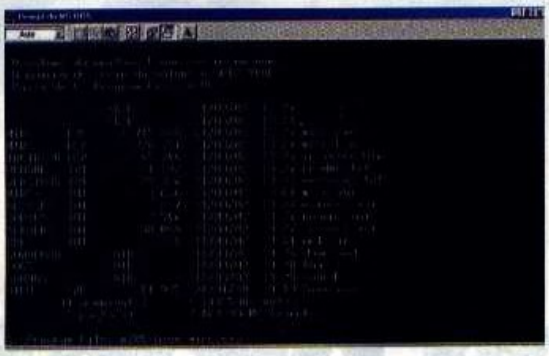
Anote o offset (endereço do código): 000C3224

Agora feche o w32dasm e vamos ao hiew

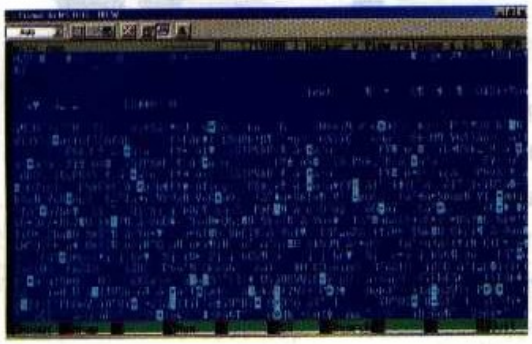
Será necessário saber um pouco de DOS, pois o hiew é um debugger que será crucial neste momento. Sua sintaxe é muito simples:

Hiew arquivo.exe

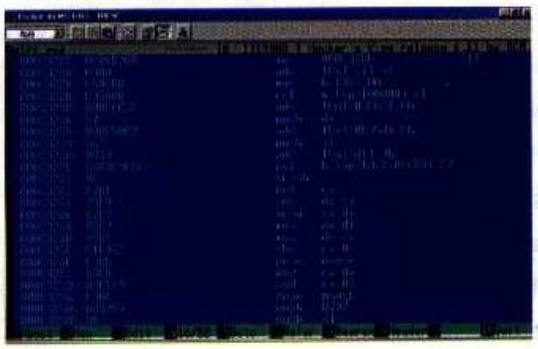
Exemplo:



Para ficar mais fácil, eu copiei o hiew para dentro do diretório do mIRC. Agora damos um *Enter* e aparece assim:



Tecle agora *F4* e, em seguida, *F2*. Agora tecle *F5* (é uma opção para procurarmos códigos na memória) e digite o código que nós encontramos: 000C3224. Se você fez certo, aparecerá assim:



Beleza. Agora é só trocar o JE por JNE. Tecle *F3* e troque o "4" de 0F84 por um "5", e ficará 0F85!! Isso que é cracking! Isso que é a quebra de programa! Pulamos o endereço do erro para cair no registro válido aceitando qualquer código!!! Agora dê um *F9* para atualizar o programa! Depois de tudo, se você fez certo, o resultado deverá ser esse:

Pronto!!! Você conseguiu quebrar um programa!!!! Pode ficar feliz! Se você fez certo, abra o mIRC de novo, vá até o menu *help*, *register*, coloque qualquer código e veja o resultado!!!

Viram que legal? Com este exemplo, é possível quebrar muitos outros programas. Mas saiba que este foi muito fácil. Existem programas com proteções mais difíceis, que requerem outras ferramentas, como o Soft Ice. Mas lembrem-se: comecem pela mensagem de erro, procurem a rotina de registro e pulem o endereço do erro para o endereço certo. JE para JNE ou vice-versa.

# Introdução à progra

C é uma linguagem fácil de aprender, especialmente se você já conhece Pascal ou alguma outra linguagem procedural. Todo conceito em Pascal leva diretamente a um conceito em C: as idéias são exatamente as mesmas, mas você usa palavras diferentes para expressá-las. O C pode parecer difícil, às vezes, porque dá mais liberdade ao programador e, portanto, é mais fácil cometer erros ou criar bugs que são difíceis de descobrir.

Este tutorial faz uma introdução ao C mostrando como fazer a conexão com o Pascal. Também introduz vários conceitos que não existem em Pascal. A maior parte desses conceitos trata dos ponteiros. Os leitores que possuem uma formação em Fortran, Cobol, BASIC, etc., perceberão como o código do Pascal é fácil de ler. Eu acredito que a única maneira de aprender C (ou qualquer outra linguagem) é escrever e ler muito código. Uma boa maneira de conseguir experiência em C é pegar programas escritos em outras linguagens e convertê-los. Dessa forma, se o programa não funcionar em C, você sabe que é a tradução que está causando o problema e não o código original. Existe uma grande diferença entre o Pascal e o C que causa vários problemas: o C não permite procedures aninhadas, assim será necessário remover todas elas para converter qualquer programa em Pascal. O melhor é que você substitua estas procedures no código Pascal. Desta forma, você pode testar novamente o programa antes de traduzi-lo para C. Também repare que o C é case sensitive. Os compiladores entendem que XXX, xxx e Xxx são três nomes diferentes. Por convenção, as constantes em C são escritas em maiúscula, enquanto as variáveis em minúsculas ou combinadas. As palavras-chave em C são sempre em minúscula.

Neste tutorial, todas as instruções de compilação e referências às páginas "man" assumem que você está trabalhando numa estação Unix normal. Se você não estiver, terá de usar os arquivos de ajuda do seu sistema para mapear as instruções para o seu ambiente.

# mação em C

## Grandes dicas para quem está começando

### Um simples programa

Abaixo temos um programa simples em C que encontra o fator de 6. Abra seu editor favorito e digite-o. Salve o programa com algum nome como exemp.c. Se você esquecer do .c, terá um erro "Bad Magic Number" quando fizer a compilação. Assim, nunca se esqueça da extensão.

```
/* Programa para encontrar o fator de 6 */
#include <stdio.h>
#define VALUE 6
int i,j;
void main()
{
j=1;
for (i=1; i<=VALUE; i++)
j=j*i;
printf("The factorial of %d is
%d\n",VALUE,j);
}
```

Na maioria dos sistemas Unix, você encontrará um programa chamado C Beautifier, que irá formatar o código para você.

Para compilar esse código, digite **cc exemp.c.** Para dar o run, digite **a.out.** Se ele não compilar ou não o fizer corretamente, edite-o novamente e veja o que deu errado.

Agora, vamos olhar o código Pascal equivalente:

```
{ Programa para encontrar o fator de 6 }
program samp;
const
value=6;
var
i,j:integer;
begin
j:=1;
for i:=1 to value do
j:=j*i;
writeln('The factorial of ',value,' is ',j);
end.
```

Você pode notar uma correspondência quase total. A única diferença real é que o código em C começa com #include <stdio.h>. Essa linha inclui a biblioteca Standard I/O no seu programa; assim você pode ler e escrever valores, trabalhar com arquivos texto e assim por diante.

C tem um número grande de bibliotecas Standard como stdio, incluindo bibliotecas de texto, tempo e matemática.

A linha #define cria uma constante. Duas variáveis globais são declaradas usando a linha int i,j;. Outros tipos de variáveis comuns são float (para números reais) e char (para caracteres), ambas podendo ser declaradas da mesma forma que int. A linha main() declara a função principal. Todo programa em C deve ter uma função chamada main em algum lugar do código. Em C, { e } substituem o begin e o end do Pascal. Da mesma forma, = substitui o operador de atribuição do Pascal :=. A estrutura de repetição for e a declaração printf são estranhas, mas elas fazem as mesmas funções que suas contrapartes no Pascal. Note que o C usa aspas duplas em vez de simples para strings.

A declaração printf em C é mais fácil de usar do que a versão Pascal, uma vez que você se acostuma com ela. A porção em aspas é chamada de format string e descreve como o dado deve ser formatado quando impresso. A format string contém string literais como The factorial of e \n para retorno de linha e operadores como marcação para variáveis. Os dois operadores na format string indicam que os valores integer encontrados mais tarde na lista de parâmetros serão colocados na string exatamente nestes pontos. Outros operadores incluem floating point, para caracteres e strings. Você pode digitar man printf para ajuda em opções de formatação.

Na declaração printf, é extremamente importante que o número de operadores na format string corresponda exatamente ao número e tipo das variáveis existentes. Por exemplo, se a format string contém três operadores, ela deve ser seguida exatamente por três parâmetros de mesmo tipo, na mesma ordem em que os especificados pelos operadores.

Esse programa é bom, mas seria melhor se pudesse ler os valores em vez de usar uma constante. Edite o arquivo, remova a constante VALUE e declare uma variável como um integer global (mudando todas as referências em minúsculas porque o valor agora é uma variável). Coloque, então, as duas linhas seguintes no começo do programa:

```
printf("Enter the value:");
scanf("%d",&value);
```

O código equivalente para isso em Pascal é:

```
write('Enter a value:');
readln(value);
```

Faça as mudanças, compile e rode o programa para ter certeza de que ele funciona. Note que scanf usa o mesmo tipo de format string que printf (digite man scanf para mais informações). Note também o sinal & na frente de value. Este é o address operator em C. Ele retorna o endereço da variável, mas isso não fará nenhum sentido até discutirmos ponteiros. Você precisa usar o operador & em scanf independente da variável ser tipo char, int ou float. Se você esquecer o operador &, você receberá um erro quando rodar o programa.

**Erros a serem evitados:**

**- Esquecer de usar o & no scanf.**

**- Muitos ou poucos parâmetros após as declarações printf ou scanf.**

**- Esquecer o * / no final dos comentários.**

## Branching e estruturas de repetição (loops)

As declarações e estruturas while em C baseiam-se nas idéias das expressões Booleanas, da mesma forma que em Pascal. Em C, entretanto, não existe o tipo Booleano: você usa integers no lugar. O integer value 0 em C é false, enquanto qualquer outro valor é true.

Aqui temos uma tradução simples do Pascal para o C. Primeiro o código em Pascal:

```
if (x=y) and (j>k) then
z:=1
else
q:=10;
```

A tradução para o C parece muito similar, mas há diferenças importantes, que nós vamos discutir em breve.

```
if ((x==y) && (j>k))
z=1;
else
q=10;
```

Perceba que = em Pascal virou == em C. Esta é uma diferença importante, porque o C irá aceitar um = simples quando você compilar, mas irá se comportar de forma diferente quando você rodar o programa. O and em Pascal vira && em C. Perceba também que z=1; em C tem um ponto e vírgula, que C elimina o then, e que a expressão Booleana precisa ser completamente cercada por parênteses. O seguinte quadro mostra a tradução de operadores Booleanos do Pascal para o C:

| Pascal | C |
|---|---|
| = | == |
| < | < |
| > | > |
| <= | <= |
| >= | >= |
| <> | != |
| AND | && |
| OR | \|\| |
| NOT | ! |

O sinal == é um problema porque é comum esquecermos e digitarmos somente =. Como os integers substituem os Booleanos, a seguinte declaração é legal em C:

```
void main()
{
int a;
printf("Enter a number:");
scanf("%d", &a);
if (a)
{
blah blah blah
}
}
```

se a for qualquer coisa diferente de 0, o código que blah blah blah representa será executado. Vejamos agora a seguinte declaração em Pascal:

```
if a=b then
```

incorretamente traduzida para C como:

```
if (a=b) /* teria que ser "if (a==b)" */
```

Em C, esta declaração significa "Atribua b para a e depois teste o valor Booleano de a". Então, se a for 0, a declaração if é falsa; senão, ela será verdadeira. O valor de a também muda. Este não é um comportamento previsto (apesar desta característica ser importante quando usada corretamente), então seja cuidadoso com as conversões entre = e ==.

As declarações while são bem fáceis de traduzir. Por exemplo, vejamos o seguinte código em Pascal:

```
while a < b do
begin
blah blah blah
end;
```

em C vira:

```
while (a < b)
{
blah blah blah
}
```

C também possui uma estrutura "do-while" para substituir o "repeat-until" do Pascal como mostrada abaixo:

```
do
{
blah blah blah
}
while (a < b);
```

A estrutura de repetição for em C é bastante diferente da sua versão em Pascal, porque a versão em C é simplesmente um atalho para expressar uma declaração while. Por exemplo, vejamos o seguinte código em C:

```
x=1;
while (x<10)
{
blah blah blah
x++; /* x++ é a mesma coisa que escrever
x=x+1. É uma adição. */
}
```

Você pode converter isso numa estrutura de repetição for da seguinte maneira:

```
for(x=1; x<10; x++)
{
blah blah blah
}
```

Note que a estrutura while contém um passo de inicialização (x=1), um passo de teste (x<10) e um passo de incremento (x++). Com a estrutura for é possível colocar qualquer coisa nestas três partes. Por exemplo, vamos dar uma olhada nesta estrutura:

```
a=1;
b=6;
while (a < b)
{
a++;
printf("%d\n",a);
}
```

Você pode colocar isso numa declaração for também:

```
for (a=1,b=6; a < b; a++,printf("%d\n",a));
```

É confuso, mas possível. A vírgula permite que você separe diferentes declarações nas seções de inicialização e incremento da estrutura for (mas não nas seções de teste). Muitos programadores em C gostam de juntar muita informação numa única linha. Eu acho que isso dificulta a compreensão do código, portanto a ordem é quebrar o código em várias linhas.

**Erros a serem evitados:**

**- Colocando = quando o certo é == numa declaração if ou while.**

**- Acidentalmente coloque um ; no final de uma estrutura de repetição ou uma declaração if, de forma que essa declaração perca o efeito. Por exemplo:**

```
for (x=1; x<10; x++);
printf("%d\n",x);
```

somente imprime um valor por causa do ponto e vírgula depois da declaração for.

## Arrays

Nesta seção, você irá criar um pequeno programa que gera dez números aleatórios e os embaralha.

Inicie o editor e digite o seguinte código:

```
#include <stdio.h>
#define MAX 10
int a[MAX];
int rand_seed=10;
int rand() /* from K&R - returns random
number between 0 and 32767.*/
{
rand_seed = rand_seed * 1103515245 +12345;
return (unsigned int)(rand_seed / 65536) %
32768;
}
void main()
{
int i,t,x,y;
/* fill array */
for (i=0; i < MAX; i++)
{
a[i]=rand();
printf("%d\n",a[i]);
}
/* more stuff will go here in a minute */
}
```

Esse código contém vários conceitos novos, mas as linhas #include e #define são familiares para você. A linha int a[MAX]; mostra como declarar um array de integers em C. Como exemplo, a declaração int a[10]; é declarado desta forma em Pascal:

```
a:array [0..9] of integer;
```

Todas arrays começam em zero e vão até n-1 em C. Pois int a[10]; contém 10 elementos e o maior índice válido é 9. Ao contrário do Pascal, C não oferece nenhuma forma de mudar os valores do índice. Notem também que por causa da posição do array a, esta situação é global para o programa inteiro.

A linha int rand_seed=10 também declara uma variável global, desta vez chamada de rand_seed que é iniciada em 10 cada vez que o programa inicia. Este valor é a onda que inicia o código de números aleatórios que se segue. Num gerador de números aleatórios real, a onda deveria iniciar num valor aleatório. Aqui, a função rand produzirá os mesmos valores a cada vez que o programa for iniciado.

A linha int rand() é uma declaração de função. A declaração de função equivalente aparece assim em Pascal:

```
function rand:integer;
```

A função rand não aceita nenhum parâmetro e retorna um valor integer.

As quatro linhas que se seguem implementam a função rand. Nós iremos ignorá-las agora.

A principal função é normal. Quatro integers locais são declaradas, e a array é preenchida com dez valores aleatórios usando uma estruturação de repetição for. Note que arrays são indexadas exatamente como em Pascal.

Agora adicione o seguinte código no lugar do comentário que começa com more stuff:

```
/* bubble sort the array */
for (x=0; x < MAX-1; x++)
for (y=0; y < MAX-x-1; y++)
if (a[y] > a[y+1])
{
t=a[y];
a[y]=a[y+1];
a[y+1]=t;
}
/* print sorted array */
printf("----------\n");
for (i=0; i < MAX; i++)
printf("%d\n",a[i]);
```

Esse código mistura os valores aleatórios e os imprime misturadamente.

**Exercícios:**

**- No primeiro pedaço do código, tente mudar a estrutura de repetição for que preenche a array, para uma linha única de código. Faça com que o resultado seja o mesmo que o código original.**

**- Inicie a onda de números aleatórios em valores diferentes.**

**Erros a evitar:**

**- C não tem checagem de série, portanto se você indexar depois do fim da array, você não será avisado. O programa pode eventualmente dar pau e retornará dados errados.**

**- Um chamado de função deve incluir () mesmo se nenhum parâmetro for passado. Por exemplo, C irá aceitar x=rand;, mas o chamado não irá funcionar. O endereço de memória da função rand será colocado em x. O correto seria x=rand();.**

## Operadores e precedência

Os operadores em C são similares aos operadores em Pascal como mostrado abaixo:

| Pascal | C |
|---|---|
| + | + |
| - | - |
| / | / |
| * | * |
| div | / |
| mod | % |

O operador / faz uma divisão integer se ambos os operadores são integers e divide os pontos flutuantes. Por exemplo:

```
void main()
{
float a;
a=10/3;
printf("%f\n",a);
}
```

Esse código imprime um valor de ponto flutuante a partir do momento em que a é declarado como um tipo flutuante, mas a será 3.0 porque o código fez uma divisão entre integers.

A precedência do operador em C é também similar ao do Pascal. Como em Pascal, os parênteses controlam a precedência.

## Typecasting

O C permite que você faça conversões de tipo facilmente. É possível fazer isso especialmente quando usando ponteiros. Typecasting também ocorre durante a operação de distribuição para certos tipos. Por exemplo, no código acima, o valor do integer foi automaticamente convertido para um ponto flutuante.

Você faz typecasting em C ao colocar o nome do tipo entre parênteses e deixando-o em frente do valor que você quer mudar. Pois, no código acima, ao trocar a linha a=10/3; por a=(float)10/3; produza 3,333333 em a porque 10 é convertido para um valor de ponto flutuante antes da divisão.

## Tipos

Você declara tipos definidos pelo usuário em C com a declaração typedef. O seguinte exemplo mostra um tipo que aparece freqüentemente no código C:

```
#define TRUE 1
#define FALSE 0
typedef int boolean;
void main()
{
boolean b;
b=FALSE;
blah blah blah
}
```

Esse código permite que você declare tipos Booleanos em programas C.

Se você não gosta da palavra "flutuante" para números reais, você pode dizer:

```
typedef float real;
```

e depois dizer:

```
real r1,r2,r3;
```

Você pode colocar declarações typedef em qualquer lugar num programa C, mas eles devem estar antes de serem usados no código. Não é necessário agrupá-los nem dê nenhuma palavra especial para marcar o início do bloco como em Pascal.

## Arrays

Você declara arrays ao inserir um tamanho de array após uma declaração normal, como mostrado abaixo:

```
int a[10]; /* array of integers */
char s[100]; /* array of characters (a C
string) */
float f[20]; /* array of reals */
struct rec r[50]; /* array of records */
```

**Incrementando**

| Versão longa | Curta versão |
|---|---|
| i=i+1; | i++; |
| i=i-1; | i--; |
| i=i+3; | i+=3; |
| i=i*j; | i*=j; |

**Erro em C a ser evitado:**

**- Como descrito acima, usando o operador / com dois integers freqüentemente produzirá um resultado inesperado, então fique bastante ligado quando for usá-lo.**

## Funções

A maioria das linguagens permite a criação de procedures e funções, ou ambos. O C permite somente funções, apesar de ser possível criar procedures criando funções que não retornam nada. As funções em C podem aceitar um número ilimitado de parâmetros. Como mencionado na introdução, elas não podem ser aninhadas. Em geral, em C não importa em que ordem você coloca suas funções no programa. Nós já falamos um pouco sobre as funções. A função rand é muito simples. Ela não aceita nenhum parâmetro e retorna um integer como resultado:

```
int rand()
/* from K&R - produces a random number
between 0 and 32767.*/
{
rand_seed = rand_seed * 1103515245 +12345;
return (unsigned int)(rand_seed / 65536) %
32768;
}
```

A linha int rand() declara a função rand para o resto do programa e especifica que rand não irá aceitar nenhum parâmetro e retornará um resultado integer. Esta função não tem nenhuma variável local, mas se ela necessitasse de locais, elas teriam que estar logo após o { de abertura. (O C na verdade permite que você declare variáveis após qualquer {. Essas variáveis desaparecem assim que o } de fechamento seja atingido. Enquanto existirem, elas ficam guardadas no sistema). Perceba que não existe nenhum ; após o () na primeira linha. Se você colocar algum acidentalmente, receberá de volta uma série grande de erros sem sentido. Note também que apesar de não ter nenhum parâmetro, você precisa usar o (), pois eles dirão ao compilador que você está declarando uma função em vez de simplesmente declarar um int.

A declaração de return é importante para qualquer função que retorna um resultado. Ela dá à função o valor para retornar e causa uma saída imediatamente. Isso significa que você pode colocar múltiplas declarações de retorno na função para conseguir múltiplos pontos de saída. Se você não colocar uma declaração de return numa função, a função retornará quando ela chegar } e dará erro (muitos compiladores avisarão se você esquecer de retornar um valor). Em C, uma função pode retornar valores de qualquer tipo.

Há diversos métodos corretos para se chamar uma função rand - por exemplo: x=rand();. O x é o valor retornado nesta declaração. Note que você precisa usar () na chamada da função, mesmo se nenhum parâmetro for passado.

Você também pode chamar a função rand desta forma:

```
if (rand() > 100)
```

Ou desta maneira:

```
rand();
```

Em último caso, o valor retornado pelo rand será descartado. Talvez você nunca queira fazer isso com rand, mas muitas funções retornam algum tipo de código de erro através do nome da função, e se você não se preocupar com códigos de erros (por exemplo, porque você sabe que um erro é impossível), você pode descartar.

Você pode criar procedures (na forma do Pascal) ao dar à função um tipo de retorno vazio. Por exemplo:

```
void print_header()
{
printf("Program Number 1\n");
printf("by Marshall Brain\n");
printf("Version 1.0, released 12/26/91\n");
}
```

Esta função não retorna nenhum valor, então é uma procedure. Você pode chamá-lo com a seguinte declaração:

```
print_header();
```

Você pode incluir () na chamada. Se você não fizer isso, a função não será chamada, apesar de ser compilado corretamente em vários sistemas.

As funções em C podem aceitar parâmetros de qualquer tipo. Por exemplo:

```
int fact(int i)
{
int j,k;
j=1;
for (k=2; k<=i; k++)
j=j*k;
return j;
}
```

Retorna o fator de i, que é passado como um parâmetro integer. Separe múltiplos parâmetros com vírgulas:

```
nt add (int i, int j)
{
return i+j;
}
```

O C avançou muito nesses anos. Você verá freqüentemente funções escritas no "velho estilo" como as que mostramos abaixo:

```
int add(i,j)
int i;
int j;
{
return i+j;
}
```

É importante que seja possível ler o código escrito no estilo antigo. Não há diferenças na sua execução; é somente uma forma diferente de se fazer a notação. É melhor usar o estilo novo, (conhecido como "ANSI C") com o tipo declarado como parte da lista de parâmetro, a menos que você vá enviar este código para alguém que só tem acesso a compiladores antigos (non-ANSI).

Agora é considerado correto usar os protótipos de função para todas as funções em seu programa. Um protótipo declara o nome da função, seu parâmetro e seu tipo de retorno para o resto do programa de uma maneira similar à declaração forward em Pascal. Para entender porque os protótipos de funções são úteis, entre com o seguinte código e dê um run:

```
#include <stdio.h>
void main()
{
printf("%d\n",add(3));
}
int add(int i, int j)
{
return i+j;
}
```

Este código compila sem lhe dar um aviso, apesar de add esperar dois parâmetros mas receber um só, porque o C não checa parâmetros. Você pode perder muito tempo debugando o código quando o problema é que você está simplesmente passando muitos ou poucos parâmetros. O código acima compila corretamente, mas ele produz uma resposta errada.

Para resolver este problema, o C permite que você coloque protótipos de funções no começo (na verdade, em qualquer lugar) do programa. Se você fizer isso, o C checa os tipos e counts de todas as listas de parâmetros. Tente compilar o seguinte código:

```
#include <stdio.h>
int add (int,int); /* function prototype for
add */
void main()
{
printf("%d\n",add(3));
}
int add(int i, int j)
{
return i+j;
}
```

O protótipo faz com que o compilador mostre um erro na declaração printf.

Coloque um protótipo para cada função no começo do seu programa. Eles podem evitar bastante tempo de debugação e podem resolver o problema que você consegue, quando compila com funções que você usa antes de eles serem declarados. Por exemplo, o seguinte código não irá compilar:

```
#include <stdio.h>
void main()
{
printf("%d\n",add(3));
}
float add(int i, int j)
{
return i+j;
}
```

Por que - você pode perguntar - ele irá compilar quando add retorna um int, mas não quando retorna um ponto flutuante?

Porque é padrão do C retornar um valor int. Usando um protótipo, você resolve este problema. Os compiladores antigos (non-ANSI) permitem protótipos, mas a lista de parâmetros para o protótipo precisa estar vazia.

Os compiladores antigos não checam os erros das listas de parâmetros.

## Bibliotecas e Makefiles

As bibliotecas são muito importantes em C porque esta linguagem só aceita as características mais básicas necessárias. O C não contém nem funções I/O para ler do teclado e escrever na tela. Qualquer coisa que ultrapasse a linguagem básica deve ser escrita por um programador. Grandes pedaços de código são colocados em bibliotecas. Nós vimos os padrões I/O, ou stdio, que também são bibliotecas. Existem outras bibliotecas para funções de matemática, manipulação de strings, manipulação de tempo e por aí vai. Bibliotecas também permitem dividir seu programa em módulos, o que torna mais fácil a compreensão.

Você pode criar suas próprias bibliotecas facilmente. Como exemplo, nós vamos pegar um código e construir uma biblioteca de duas de suas funções:

```
#include <stdio.h>
#define MAX 10
int a[MAX];
int rand_seed=10;
int rand()
/* from K&R - produces a random number
between 0 and 32767.*/
{
rand_seed = rand_seed * 1103515245 +12345;
return (unsigned int)(rand_seed / 65536) %
32768;
}
void main()
{
int i,t,x,y;
/* fill array */
for (i=0; i < MAX; i++)
{
a[i]=rand();
printf("%d\n",a[i]);
}
/* bubble sort the array */
for (x=0; x < MAX-1; x++)
for (y=0; y < MAX-x-1; y++)
if (a[y] > a[y+1])
{
t=a[y];
a[y]=a[y+1];
a[y+1]=t;
}
/* print sorted array */
printf("----------\n");
for (i=0; i < MAX; i++)
printf("%d\n",a[i]);
}
```

Esse código preenche uma array com números aleatórios, mistura-os com o bubble sort e mostra a lista misturada.

Pegue o código bubble sort e use o que você já aprendeu para construir uma função. Já que tanto o array a e a constante MAX são conhecidas globalmente, a função que você cria não necessita de nenhum parâmetro, nem precisa retornar um resultado. Entretanto, você deveria usar variáveis locais para x, y e t.

Uma vez que você testou a função para ter certeza que está trabalhando, coloque o número de elementos como um parâmetro em vez de usar MAX. Faça isso primeiro sem olhar para o código abaixo e então compare os dois somente quando você tiver terminado.

```
#include <stdio.h>
#define MAX 10
int a[MAX];
int rand_seed=10;
int rand() /* from K&R - returns random
number between 0 and 32767.*/
{
rand_seed = rand_seed * 1103515245 +12345;
return (unsigned int)(rand_seed / 65536) %
32768;
}
void bubble_sort(int m)
{
int x,y,t;
for (x=0; x < m-1; x++)
for (y=0; y < m-x-1; y++)
if (a[y] > a[y+1])
{
t=a[y];
a[y]=a[y+1];
a[y+1]=t;
}
}
void main()
{
int i,t,x,y;
/* fill array */
for (i=0; i < MAX; i++)
```

```
{
a[i]=rand();
printf("%d\n",a[i]);
}
bubble_sort(MAX);
/* print sorted array */
printf("----------\n");
for (i=0; i < MAX; i++)
printf("%d\n",a[i]);
}
```

Você também pode generalizar a função bubble_sort ainda mais ao tornar **a** e o tamanho de **a** um parâmetro:

```
bubble_sort(int m, int a[])
```

Esta linha diz, "Aceita o array integer a de qualquer tamanho como um parâmetro". Nada no corpo da função bubble_sort precisa mudar. Para chamar a bubble_sort mude o chamado para:

```
bubble_sort(MAX,a);
```

Note que **&a** não foi usada apesar de que a mistura irá mudar **a.**

## Criando bibliotecas

Como as funções rand e bubble_sort no programa acima são úteis, você provavelmente vai querer reusá-los em outros programas que você escrever.

Você pode colocá-los numa biblioteca de utilitários para facilitar sua vida.

Toda biblioteca consiste em duas partes: um arquivo header e o arquivo com o código propriamente dito. O arquivo header, normalmente salvo com um sufixo .h, contém informações sobre a biblioteca que são necessárias para os programas que estão sendo usados nestas bibliotecas. Em geral, o arquivo header contém tipos e constantes, junto com headers para funções encontradas na biblioteca. Digite o seguinte arquivo header e salve-o como um arquivo chamado **util.h.**

```
/* util.h */
extern int rand();
extern void bubble_sort(int,int []);
```

Essas duas linhas lembram os protótipos de funções. A palavra "extern" em C representa funções que serão linkados depois. Num compilador antigo, remova os parâmetros da lista do bubble_sort.

Digite o seguinte código num arquivo chamado util.c.

```
/* util.c */
#include "util.h"
int rand_seed=10;
int rand()
/* from K&R - produces a random number
between 0 and 32767.*/
{
rand_seed = rand_seed * 1103515245 +12345;
return (unsigned int)(rand_seed / 65536) %
32768;
}
void bubble_sort(int m,int a[])
{
int x,y,t;
for (x=0; x < m-1; x++)
for (y=0; y < m-x-1; y++)
if (a[y] > a[y+1])
{
t=a[y];
a[y]=a[y+1];
a[y+1]=t;
}
}
```

Note que o arquivo inclui seu próprio arquivo header (util.h) e que usa aspas em vez dos símbolos < e >, que são usados somente para bibliotecas de sistema. Como você pode ver, isso é normal no código C. Veja também que a variável rand_seed por não estar no arquivo header, não pode ser visto ou modificado por um programa usando esta biblioteca. Chamamos isto de ocultação de informação. Adicionando a palavra static na frente de int faz isso.

Digite o seguinte programa e salve-o como main.c.

```
#include <stdio.h>
#include "util.h"
#define MAX 10
int a[MAX];
void main()
{
int i,t,x,y;
/* fill array */
for (i=0; i < MAX; i++)
{
a[i]=rand();
printf("%d\n",a[i]);
}
bubble_sort(MAX,a);
/* print sorted array */
printf("----------\n");
for (i=0; i < MAX; i++)
printf("%d\n",a[i]);
}
```

## Compilando e rodando uma Biblioteca

Para compilar uma biblioteca, digite o seguinte comando (assumindo que você está usando um Unix):

```
cc -c -g util.c
```

O -c leva o compilador a produzir um arquivo de objeto para a biblioteca. O arquivo contém o código de máquina da biblioteca. Ele não pode ser executado até ser linkado a um arquivo de programa que contenha a função main. O código de máquina reside num arquivo separado chamado util.o.

Para compilar o programa principal, digite o seguinte:

```
cc -c -g main.c
```

Essa linha cria um arquivo chamado main.o, que contém o código de máquina para o programa principal.

Para criar um arquivo executável que contenha o código de máquina para o programa inteiro, faça o link de dois arquivos de objetos digitando o seguinte:

```
cc -o main main.o util.o
```

Linka main.o e util.o para formar um executável chamado main. Para rodá-lo, digite main.

Pode ser extremamente incômodo digitar todas as linhas cc todas as vezes, especialmente se você estiver fazendo diversas mudanças no código e ele tiver diversas bibliotecas. A facilidade make resolve este problema. Você pode usar o seguinte makefile para substituir a seqüência de compilação acima:

```
main: main.o util.o
cc -o main main.o util.o
main.o: main.c util.h
cc -c -g main.c
util.o: util.c util.h
cc -c -g util.c
```

Insira isso num arquivo chamado makefile e digite make para construir o executável. Note que você precisa preceder todas as linhas cc com um tab. (Oito espaços não são suficientes - é necessário usar tab).

Este makefile contém dois tipos de linhas. As linhas alinhadas à esquerda são linhas de dependência. As linhas precedidas por um tab são executáveis, que podem conter qualquer comando Unix válido. Uma linha de dependência diz que algum arquivo é dependente de algum outro arquivo. Por exemplo, main.o: main.c util.h diz que o arquivo main.o é dependente dos arquivos main.c e util.h.

Se ambos os arquivos mudarem, a seguinte linha executável deve ser executada para recriar main.o.

Note que o executável final produzido pelo makefile inteiro é main, na linha 1 do makefile. O resultado final do makefile deveria ir sempre na primeira linha, que é onde o makefile diz que o arquivo main é dependente do main.o e util.o.

É possível colocar linhas múltiplas para serem executadas abaixo de uma linha de dependência - todas elas devem começar com um tab. Um programa comprido pode ter várias bibliotecas e um programa main. O makefile recompila automaticamente tudo que precisa ser recompilado por causa das mudanças.

## Arquivos texto em C

Os arquivos texto em C são retos e fáceis de entender. Eles trabalham da mesma forma que os arquivos de texto em Pascal. Todas as funções e tipos dos arquivos de texto em C vêm da biblioteca stdio.

Quando você precisa do texto I/O num programa C, precisa somente de uma fonte para incluir informação e um repositório para saída de informação. Você pode usar stdin (standard in) e stdout (standard out). Então, você usa o redirecionamento input e output na linha de comando para mover diferentes pedaços de informação através do programa. Há seis diferentes comandos I/O em <stdio.h> que você pode usar com stdin e stdout:

**printf** – imprime output formatada para stdout.

**scanf** – lê input formatado para stdin.

**puts** – imprime uma string para stdout.

**gets** – lê uma string de stdin.

**putc** – imprime um caracter para stdout.

**getc, getchar** – lê um caracter de stdin.

A vantagem de stdin e stdout é que eles são fáceis de usar. Por isso, a habilidade de redirecionar I/O é muito poderosa. Por exemplo, pode ser que você queira criar um programa que leia de stdin e conta o número de caracteres:

```
#include <stdio.h>
#include <string.h>
void main()
{
char s[1000];
int count=0;
while (gets(s))
count += strlen(s);
printf("%d\n",count);
}
```

Entre com esse código e rode-o. Ele espera por uma entrada do stdin, então digite umas poucas linhas. Quando estiver pronto, pressione Ctrl-D para assinalar um end-of-file (eof). gets lê uma linha até

detectar eof, então retorna a 0 de forma que a repetição while termina. Quando você pressiona Ctrl-D, você vê a contagem do número de caracteres em stdout (a tela).

Agora, suponha que você queira contar os caracteres num arquivo. Se você compilou o programa para a.out, você pode digitar o seguinte:

```
a.out < filename
```

Em vez de aceitar a entrada de dado do teclado, o conteúdo do arquivo nomeado filename será usado. Você pode conseguir o mesmo resultado usando pipes:

```
cat < filename | a.out
```

Você também pode redirecionar a saída para um arquivo:

```
a.out < filename > out
```

Esse comando coloca o contador de caracteres produzido pelo programa num arquivo de texto chamado out. Às vezes, você precisa usar um arquivo de texto diretamente. Por exemplo, você poderia precisar abrir um nome de arquivo específico e ler ou escrever. Você pode querer administrar diversos pedaços de entrada ou saída, ou criar um programa como um editor de texto que pode salvar e recuperar os dados ou os arquivos de configuração.

**fopen** – abre um arquivo de texto

**fclose** – fecha um arquivo de texto

**feof** – detecta uma marca de fim de arquivo

**fprintf** – imprime uma saída formatada para um arquivo

**fscanf** – lê uma entrada formatada de um arquivo

**fputs** – imprime uma string para um arquivo

**fgets** – lê uma string de um arquivo

**fputc** – imprime um caractere para um arquivo

**fgetc** – lê um caractere de um arquivo

Você usa fopen como reset e rewrite em Pascal. Isso abre um arquivo para um módulo específico (os três mais comuns são r, w e a, para ler, escrever e adicionar). Ele então retorna um ponteiro de arquivo que você usa para acessar o arquivo. Por exemplo, suponha que você queira abrir um arquivo e escrever os números de 1 a 10 nele. Você poderia usar o seguinte código:

```
#include <stdio.h>
#define MAX 10
void main()
{
FILE *f;
int x;
f=fopen("out","w");
for(x=1; x<=MAX; x++)
fprintf(f,"%d\n",x);
fclose(f);
}
```

A declaração fopen aqui abre um arquivo chamado out com o módulo w. Este é um modo de escrita destrutivo, o que significa que se o out não existe, ele é criado, mas se ele já existe, é destruído e um novo arquivo é criado em seu lugar. O comando fopen retorna um ponteiro para o arquivo, que fica guardado na variável f. Esta variável é usada para se referir ao arquivo. Se este arquivo não pode ser aberto por alguma razão, f irá conter NULL.

A declaração fprintf deve parecer bastante familiar. É igual a printf, mas usa um arquivo ponteiro como seu primeiro parâmetro. A declaração fclose fecha o arquivo quando você terminar.

Para ler um arquivo, abra-o com o módulo r. Em geral, não é uma boa idéia usar fscanf para ler. A não ser que o arquivo esteja perfeitamente formatado, fscanf não irá tratá-lo corretamente. No lugar, use fgets para ler cada linha e faça o parse dos pedaços necessários.

O seguinte código demonstra o processo de leitura de um arquivo e apresenta seu conteúdo para a tela:

```
#include <stdio.h>
void main()
{
FILE *f;
char s[1000];
f=fopen("infile","r");
while (fgets(s,1000,f)!=NULL)
printf("%s",s);
fclose(f);
}
```

A declaração fgets retorna um valor NULL na marca do fim do arquivo. Ele lê uma linha (até 1.000 caracteres neste caso) e então imprime para stdout. Perceba que a declaração printf não inclui \n no format string, porque fgets adiciona \n ao final de cada linha que lê. Por isso, você pode descobrir se uma linha não é completa quando ele ultrapassa o limite máximo especificado no segundo parâmetro do fgets.

**Erro a ser evitado:**

**- Não digite acidentalmente close no lugar de fclose. A função close existe, por isso o compilador aceita. O programa até trabalhará direito se ele só abrir ou fechar alguns arquivos. Entretanto, se o programa abrir e fechar um arquivo numa estrutura de repetição, ele eventualmente ficará sem capacidade e/ou memória, porque close não está fechando os arquivos corretamente.**

# Object Pascal

## Um pequeno tutorial sobre a linguagem que originou e é a base do Delphi

Para facilitar o uso do Delphi, é importante conhecer a linguagem Pascal. Aliás, o Delphi é uma derivação do Pascal, podemos citar outra que é o Object Pascal. Ambas são orientadas a objeto e significam um avanço importante para os programadores.

O Object Pascal é uma extensão do Pascal que foi desenvolvida pela Apple em consulta com o próprio inventor da linguagem, Niklaus Wirth. Ela descende de uma primeira tentativa de construir uma versão orientada a objetos do Pascal, o Clascal. Existem poucas diferenças sintáticas entre o Pascal e o Object Pascal.

Neste artigo, apresentamos alguns conceitos básicos do Object Pascal, pois como ele é a base de toda a programação Delphi, isso ajudará a melhorar seu aprendizado.

A linguagem Object Pascal contém todos os conceitos da orientação a objetos em geral, como encapsulamento, herança e polimorfismo (veja a matéria sobre *Programação Orientada a Objeto* para uma melhor compreensão destes conceitos). Várias extensões foram incluídas para facilitar o uso. Podemos citar os conceitos de propriedades (particulares e públicas) e tipos de informações em modo run-time e referências de classe.

### Palavras reservadas

Existem diversas palavras que são reservadas para comandos específicos do programa e não devem ser usadas para variáveis, por exemplo, nem redefinidas.

| | | | |
|---|---|---|---|
| And | Exports | Library | Set |
| Array | File | Mod | Shl |
| As | Finnaly | Nil | Shr |
| Asm | For | Not | String |
| Begin | Function | Object | Then |
| Case | Goto | Of | To |
| Class | If | On | Try |
| Const | Implementation | Or | Type |
| Constructor | In | Packed | Unit |
| Destructor | Inherited | Procedure | Until |
| Div | Initialization | Program | Uses |
| Do | Inline | Property | Var |
| Downto | Interface | Raise | While |
| Else | Is | Record | With |
| End | Label | Repeat | Xor |
| Except | | | |

### Números

Podemos definir variáveis e constantes de tipos Inteiro ou Real usando qualquer número decimal (de 0 a 9), mas a linguagem Object Pascal aceita as notações hexadecimal e científica.

### Constantes

A definição de constante é simples: é tudo aquilo que

é fixo ou estável. Aqui, ela identifica os valores fixos do programa. A sintaxe é:

```
[Declaração Constante]  [Identificador]  (=)
[constante] (;)
```

Veja, na tabela abaixo, algumas funções que podem ser utilizadas para a declaração de constantes:

| | | | |
|---|---|---|---|
| Ab | Length | Ord | SizeOf |
| Chr | Lo | Pred | Succ |
| Hi | Low | Ptr | Swap |
| High | Odd | Round | Trunc |

Alguns exemplos para a definição de Constantes:

```
const Min = 0;
Max = 100;
Centro = (Max - Min) div 2;
Beta = Chr(225);
NumLetras = Ord('Z') - Ord('A') + 1;
MensOla = 'Instrução inválida';
MensErro = ' Erro: ' + MensOla + '. ';
PosErr = 80 - Length(MensErro) div 2;
Ln10 = 2.302585092994045684;
Ln10R = 1 / Ln10;
DigNumericos = ['0'..'9'];
LetrasAlpha = ['A'..'Z', 'a'..'z'];
AlphaNum = LetrasAlpha + DigNumericos;
```

## Expressões

Como em todas as linguagens, as expressões em Object Pascal são formadas por operadores e operandos, que podem ser divididas em quatro categorias básicas:

| | |
|---|---|
| **Únicos** | @, Not |
| **Multiplicativos** | >, /, div, mod, and, shl, shr, as |
| **Adicionais** | +, -, or, xor |
| **Relacionais** | =, <>, <, >, <=, >=, in, is |

## Identificadores

Identificadores servem, é claro, para identificar todos os tipos de componentes, constantes, tipos, variáveis, procedures, etc. Há algumas pequenas regras para se criar um identificador, vejamos as duas principais:

- ele deve ser iniciado por uma letra ou caractere underscore;
- não é permitida a utilização de espaços para a formação do nome.

Vejamos os exemplos usados no nosso tutorial da página 18: btn1, edtval e por aí vai.

## Declarações

As declarações servem para descrever as ações a serem executadas por um algoritmo.

### with... do...;

Delimita um determinado bloco de declarações para um identificador específico evitando a declaração deste identificador. A sintaxe do comando é: WITH {nome do identificador} DO {comandos};. Ex:

```
begin
    { ... comandos iniciais ... }
    with form1 do
    begin
       Caption := 'Teste';          ò Equi-
valente a Form1.Caption
       BorderStyle := bsSizable;    ò Equi-
valente a Form1.BorderStyle
    end;
end;
```

### array [ ... ] of ...;

Define um conjunto de variáveis ou constantes de um mesmo tipo. A sintaxe do comando é: array [{quantidade de ocorrencias}] of {Tipo};. Os arrays são controlados por três funções:

| Função | Valor de Retorno |
|---|---|
| **Low** | Primeiro elemento |
| **High** | Aponta para o último elemento |
| **SizeOf** | Tamanho do array |

Ex:

```
const
   t: array [1..50] of Char    { Declara 50
elementos para o tipo Char }
var
   s : array[1..100] of Real   { Declara 100
elementos para o tipo real }
  ind: Integer;
begin
   for Ind := Low(s) to High(s) do s[Ind] :=
0;     { Zera os elementos do array S }
```

```
    if SizeOf(t) = 'C' then exit;
        { Se o último elemento do array T
for 'C' sai do bloco }
    { ... outros comandos... }
end;
```

### begin... end;

Restringe um conjunto de declarações dentro de um bloco de comandos determinado. Vejamos sua sintaxe:

```
BEGIN {comandos} END;
```

Ex:

```
begin
    { ... comandos iniciais... }
    begin
       { ... bloco 1 ... }
    end;
    begin
       { ... bloco 2 ... }
    end;
    { ... comandos finais ... }
end;
```

### if... then... else...;

Usada quando devemos escolher entre o resultado de uma condição booleana. Um caminho verdadeiro (then) ou outro falso (else). A sintaxe do comando é:

```
IF {condição} THEN {bloco de comandos} ELSE
{bloco de comandos};.
```

Ex:

```
begin
    { ... comandos iniciais ... }
    if  x = 3 then
       { ... Bloco verdadeiro ... }
    else
       { ... Bloco falso ... };
end;
```

### goto... ;

Envia a execução do programa para o ponto que é determinado pelo Label. A sintaxe do comando é:

```
GOTO {Label};.
```

Ex:

```
label
   primeiro;
begin
    { ... comandos iniciais ... }
    if x = 2 then
       goto primeiro;
    { ... outros comandos ... }
Primeiro:
    { ... comandos do Primeiro ... }
end;
```

### case... of... else... end;

É uma lista de declarações que satisfaz a condição de um seletor de expressões, se nenhuma parte da lista chega a satisfazer estas condições, o subcomando else é executado. A sintaxe do comando é:

```
CASE {seletor} OF {Expressão 1}: {comando da
expressão 1}; {Expressão 2}: {comando da
expressão 2}; {Expressão n}: {comando da
expressão n} ELSE {comando}; end;.
```

Ex:

```
begin
    { ... comandos iniciais ... }
    case x of
       1: { ... Bloco para x = 1 ... }
       2, 3: { ... Bloco para x = 2 ou X =
3... }
       4..6: { ... Bloco para  4 <= x <= 6
... }
    else
       { ... Bloco para  x < 1 ou x > 6 ...
};
    end;
end;
```

### repeat... until;

Repete um bloco de declarações até a satisfação da condição booleana que está no subcomando until. A sintaxe do comando é:

```
REPEAT {comandos}; until {condição};.
```

Ex:

```
begin
    { ... comandos iniciais ... }
    x := 0;
    repeat
       x := x + 1
    until (x = 2);
end;
```

### for... to (downto)... do...;

Acrescenta em 1 (um) uma variável inteira, repetindo o bloco de comandos, até que seja atingido o valor final do intervalo. O subcomando downto realiza o reverso. A sintaxe do comando é:

```
FOR {variavel} := {valor inicial} to (downto)
{valor final} do {bloco de comandos};.
```

Ex:

```
begin
     { ... comandos iniciais ... }
     for i := 1 to 10 do          ò Executa o
[comandos A] para i = 1,2,3,4,5,6,7,8,9 e 10
        { ... Comandos A ... }
     for s := 10 downto 1 do      ò Executa o
[comandos B] para i = 10,9,8,7,6,5,4,3,2 e 1
        { ... Comandos B... }
end;
```

### while... do...;

O bloco de comandos será repetido enquanto determinada condição booleana for satisfeita. A sintaxe do comando é:

```
WHILE {condição} DO {bloco de comandos};.
```

Ex:

```
begin
     { ... comandos iniciais ... }
     while i := 1 do          ò Repete o [Bloco
de comandos] enquanto i = 1
        { ... Bloco de comandos ... }
end;
```

### break; ou continue...;

O comando break interrompe um bloco de repetição for, while ou repeat saindo do bloco. A sintaxe do comando é:

```
BREAK;
```

enquanto que o comando continue retorna a primeira instrução do bloco de repetição for, while ou repeat. A sintaxe do comando é:

```
CONTINUE;.
```

Ex:

```
begin
    { ... comandos iniciais ... }
    for i := 1 to 10 do
    begin
        if i = 8 then
            break;              ò Salta para
os [comandos C]
        {... comandos A...}
        if i = 5 then
            continue;           ò Retorna
para o comando for pulando os [comandos B]
        {... comandos B ...}
    end;
    {... comandos C ...}
end;
```

## Blocos de Procedimentos ou Funções

As procedures são declaradas na seção de tipos de declarações (abaixo do comando type) pertencendo ao objeto. Elas podem ser do tipo public (públicas - executadas por outras unidades) ou private (particulares - restritas à unidade local).

### Procedure

```
procedure {cabeçalho}; var {declaração das
variáveis}; {bloco de comandos};
```

O cabeçalho da procedure é composto pelo nome do procedimento e variáveis que serão recebidas (ou modificadas) através da declaração var, ex: `procedure teste(var x:string);`.

```
procedure soma(a,b: integer);          ò Início
enviando as variáveis A e B do tipo inteiro.
var                                    ò Declaração de
variáveis locais.
  c: integer;
begin                                  ò Corpo do proce-
dimento.
  c := a + b;
end;
```

### Function

```
function {cabeçalho} : {resultado}; var {de-
claração das variáveis}; {bloco de comandos};
```

As funções se diferem dos procedimentos pela obrigatoriedade do retorno de um resultado, podendo este resultado ser retornado pela declaração: `{nome da função} := valor` ou `result := valor`.

```
function soma(a,b: integer) : integer;     ò
Início enviando as variáveis A e B do tipo
inteiro.
begin                                  ò Corpo do proce-
dimento.
  soma := a + b;                             ò ou
result := a + b;
end;
```

Este foi um pequeno artigo para apresentar conceitos básicos sobre o Object Pascal. Ele não substitui o estudo com afinco da linguagem, apenas dá uma noção e ajuda na construção da pequena calculadora que está na página 18. O que vimos aqui não é exclusivo do Object Pascal, mas é comum a várias linguagens de programação. Agora, vamos entender as características principais da Linguagem Orientada a Objetos.

# Dicas de Delphi

## >>> Reunimos abaixo 50 dicas importantes que facilitarão seu aprendizado da linguagem Delphi.

Marcelo Jaloto Machado
mjaloto@bol.com.br
www.jaloto.rg3.net

## 1) MOSTRAR E ESCONDER O BOTÃO INICIAR

Crie um *subdiretório* chamado "**Botão Iniciar**" utilizando o *Windows Explorer*.

Depois abra o Delphi; feche o projeto que estiver aberto usando a opção *Close all* dentro do menu *File* e crie um novo projeto utilizando a opção *New Application* também no menu *File*.

**a)** Mude as seguintes propriedades do Form1:

Name: *frmEsconderMostrar*

Caption: *programa para Esconder e Mostrar o Botão Iniciar*

Position: *poScreenCenter*

BorderStyle: *bsDialog*

Height: *104*

Width: *403*

**b)** Insira dois Botões no formulário: na Paleta de Componentes *Standard - Button*

**c)** Mude as seguintes propriedades do Button1:

Name: *btnEsconder*

Caption: *esconder o Botão Iniciar*

Width: *177*

**d)** Mude as seguintes propriedades do Button2:

Name: *btnMostrar*

Caption: *mostrar o Botão Iniciar*

Width: *177*

**OBS.: Salve o projeto no *subdiretório* que você criou:**

**e)** A *Unit1* salve com o nome de *untEsconderMostrar*, e o *Project1* com o nome de *EsconderMostrar*

**f)** Na parte interface da unit (untEsconderMostrar), abaixo da cláusula uses, inclua a definição da procedure

```
interface
uses
  Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
  StdCtrls;
  Procedure
MostrarEsconderIniciar(Estado:Boolean); {in
clua esta linha}
```

Na parte implementation da unit (untEsconderMostrar) inclua a procedure MostrarEsconderIniciar:

```
implementation

{$R *.DFM}

procedure
MostrarEsconderIniciar(Estado:Boolean);
Var taskbarhandle, buttonhandle : HWND;
begin
taskbarhandle := FindWindow('Shell_TrayWnd',
nil);
buttonhandle := GetWindow(taskbarhandle,
GW_CHILD);
If Estado = True Then
          ShowWindow(buttonhandle,
SW_RESTORE) {mostra o botão}
Else
          ShowWindow(buttonhandle, SW_HIDE);
{esconde o botão}
end;
```

## 2) MOSTRAR E ESCONDER A BARRA DE TAREFAS

Crie um *subdiretório* chamado "**Barra de Tarefa**" utilizando o *Windows Explorer*.

Feche o projeto que estiver aberto usando a opção *Close all* dentro do menu *File* e crie um novo projeto utilizando a opção *New Application* também no menu *File*.

**a)** Mude as seguintes propriedades do Form1:

Name: *frmBarraTarefa*

Caption: *programa para esconder e mostrar a Barra de Tarefa*

Position: *poScreenCenter*

BorderStyle: *bsDialog*

Height: *104*

Width: *403*

**b)** Insira *dois* Botões no formulário: na Paleta de Componentes *Standard - Button*

**c)** Mude as seguintes propriedades do Button1:

Name: *btnEsconder*

Caption: *esconder a Barra de Tarefa*

Width: *177*

**d)** Mude as seguintes propriedades do Button2:

Name: *btnMostrar*

Caption: *mostrar a Barra de Tarefa*

Width: *177*

**OBS.: Salve o projeto no *subdiretório* que você criou:**

**e)** A *Unit1* salve com o nome de *untBarraTarefa* e o *Project1* com o nome de *BarraTarefa*.

**f)** Na parte interface da unit (untBarraTarefa) abaixo da cláusula uses inclua a definição da procedure

```
interface

uses
  Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
  StdCtrls;

 procedure EscondeBarraTarefa(EstadoBarra:
Boolean);{inclua esta linha}
```

Na parte implementation da unit (untBarraTarefa) inclua a procedure EscondeBarraTarefa:

```
implementation

{$R *.DFM}

procedure EscondeBarraTarefa(EstadoBarra:
Boolean);
var wndHandle : THandle;
      wndClass : array[0..50] of Char;
begin
      StrPCopy(@wndClass[0],'Shell_TrayWnd');
      wndHandle := FindWindow(@wndClass[0],
nil);
      If EstadoBarra=True Then
            ShowWindow(wndHandle, SW_RESTORE)
{Mostra a barra de tarefas}
            Else
            ShowWindow(wndHandle, SW_HIDE);
{Esconde a barra de tarefas}
end;
```

## 3) PEGANDO O NOME DO USUÁRIO E A EMPRESA DO WINDOWS

Crie um *subdiretório* chamado "**Usuário**" utilizando o *Windows Explorer*.

Feche o projeto que estiver aberto usando a opção *Close all* dentro do menu *File* e crie um novo projeto utilizando a opção *New Application* também no menu *File*.

**a)** Mude as seguintes propriedades do Form1:

Name: *frmEmpresausuario*

Caption: *programa para ler do Windows nome do usário e empresa*

Position: *poScreenCenter*

BorderStyle: *bsDialog*

Height: *123*

Width: *441*

**b)** Insira *um* Botão no formulário: na Paleta de Componentes *Standard - Button*

**c)** Mude as seguintes propriedades do Button1:

Name: *btnUsuario*

Caption: *pegar nome do usuário e empresa no Windows*

Width: *241*

**d)** Insira *duas* Caixas de Edição no formulário: na Paleta de Componentes *Standard - Edit*

**e)** Mude as seguintes propriedades do Edit1:

Name: *EdtUsuario*

Text: *vazio*

Width: 417

**f)** Mude as seguintes propriedades do Edit2:

Name: *EdtEmpresa*

Text: *vazio*

Width: 417

**OBS.: Salve o projeto no *subdiretório* que você criou:**

**g)** A *Unit1* salve com o nome de *untEmpresaUsuario*, e o *Project1* com o nome de *Usuario*.

**h)** Na parte *uses* da interface da unit (*untEmpresaUsuario*) insira a cláusula: *Registry*

```
interface

uses
     Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
StdCtrls, Registry;
```

**No evento onClick do botão btnUsuario inclua as seguintes linhas de código:**

```
procedure TForm1.Button1Click(Sender:
TObject);
var
reg: TRegIniFile;
begin
      reg :=
TRegIniFile.create('SOFTWARE\MICROSOFT\MS
SETUP (ACME)\');
     EdtUsuario.Text := reg.ReadString('USER
INFO','DefName','');
     EdtEmpresa.Text := reg.ReadString('USER
INFO','DefCompany','');
     reg.free;
end;
```

## 4) COMO ARRASTAR UM FORM SEM CLICAR NO CAPTION?

Quando você pressiona o botão do mouse, o Windows identifica a posição da tela onde o cursor estava no momento do clique. Se a posição é igual a área do Caption do Form, o Windows ativa o modo de movimentação do Form permitindo que este seja arrastado. Portanto, a maneira mais fácil de solucionar esta questão é "enganar" o Windows.

Neste exemplo, vamos considerar que o usuário poderá arrastar o Form ao clicar na área cliente deste Form:

**a)** Crie uma nova aplicação;

**b)** Adicione a seguinte declaração na seção private do Form:

*procedure* WMNCHitTest(var M: TWMNCHitTest); *message* wm_NCHitTest;

**c)** Adicione o código deste procedimento na seção *implementation* do *Form*:

```
procedure TForm1.WMNCHitTest(var M:
TWMNCHitTest);
begin
inherited;              { ativa a herança da
mensagem }
if M.Result = htClient then          { o
clique foi na área cliente? }
     M.Result := htCaption;     { se sim, faz
o Windows pensar que foi no Caption. }
     end;
```

Este exemplo tratou o clique na área cliente. Você pode alterar este código para suas necessidades. Eis os possíveis valores para o **Result**:

VALOR - Local do clique

HTBORDER - Borda da janela que não tem a borda de tamanho

HTBOTTOM - Borda horizontal inferior da janela

HTBOTTOMLEFT - Canto inferior esquerdo da janela

HTBOTTOMRIGHT - Canto inferior direito da janela

HTCAPTION - Barra de Título(Caption)

HTCLIENT - Área cliente

HTERROR - igual ao HTNOWHERE, a diferença é que produz um beep indicando erro

HTGROWBOX - Caixa de tamanho (igual ao HTSIZE)

HTHSCROLL - Barra de rolagem horizontal

HTLEFT - Borda esquerda da janela

HTMENU - Em um menu

HTNOWHERE - Plano de fundo da janela ou linha de divisão entre janelas

HTREDUCE - Botão minimizar

HTRIGHT - Borda direita da janela

HTSIZE - Caixa de tamanho (igual ao HTGROWBOX)

HTSYSMENU - Botão de Sistema/Fechar da janela MDIChild

HTTOP - Borda horizontal superior da janela

HTTOPLEFT - Canto superior esquerdo da janela

HTTOPRIGHT - Canto direito superior da janela

HTTRANSPARENT - Janela em segundo plano

HTVSCROLL - Barra de rolagem vertical

HTZOOM - Botão maximizar

## 5) BLOQUEAR A TECLA CTRL+DEL DO DBGRID.

```
procedure TForm1.DBGrid1KeyDown(Sender:
TObject; var Key: Word; Shift: TShiftState);
begin
if ((Shift = [ssCtrl]) and (key = vk_delete))
THEN
Abort;
end;
```

## 6) ESCONDENDO A APLICAÇÃO DA BARRA DE TAREFAS

Para fazer com que o ícone da aplicação em Delphi desapareça da Barra de Tarefas, execute o código a seguir:

```
var
H : HWnd;
begin
H := FindWindow(Nil,'Project1');
if H <> 0 then ShowWindow(H,SW_HIDE);
end;
```

## 7) OS COMANDOS INC E DEC

Você sabia que existe uma opção para a comum linha de comando: Variavel:=Variavel+1; ?

Os comandos INC e DEC permitem agilizar o processamento do seu sistema. Para isso, substitua a linha acima por:

```
INC(variavel);
```

ou

DEC(variavel) se você quiser diminuir ao invés de aumentar 1.

## 8) COMO FAZER UM BEEP NO COMPUTADOR

```
messageBeep(0);
```

## 9) COMO FAZER UMA PAUSA POR UM PERÍODO DETERMINADO

NumSec é o tempo em segundos de espera

```
var
NumSec SmallInt;
StartTime: TDateTime;
begin
StartTime := now;
NumSec:=10;
repeat
Application.ProcessMessages;
until Now > StartTime + NumSec * (1/24/60/
60);
end;
```

## 10) DESABILITANDO O SPLASH SCREEN DO REPORT SMITH

1 - Localize o arquivo RS_RUN.INI (no diretório do Windows);

2 - Na seção [ReportSmith] inclua a linha seguinte:

ShowAboutBox=0

3 - Na seção [RS_RunTime] inclua a linha seguinte:

ShowAboutBox=0

4 - Não se esqueça de distribuir com o seu aplicativo o referido arquivo INI.

## 11) LENDO O VOLUME DO HD

```
Function
ExtractDiskSerial(Drive:String):String;
Var
Serial:DWord;
DirLen,Flags: DWord;
DLabel : Array[0..11] of Char;
begin
GetVolumeInformation(PChar(Drive+':\'),
dLabel,12,@Serial,DirLen,Flags,nil,0);
Result := IntToHex(Serial,8);
end;
```

## 12) DESCOBRINDO O NÚMERO SERIAL DO HD

```
 procedure TForm1.Button1Click(Sender:
TObject);
 var
     SerialNum : pdword;
     a, b : dword;
```

```
    Buffer  : array [0..255] of char;
begin
    if GetVolumeInformation('c:\', Buffer,
SizeOf(Buffer), SerialNum, a,.b, nil, 0) then
        Label1.Caption :=
IntToStr(SerialNum^);
  end;
```

## 13) PARA SABER SOMENTE O PATH DA APLICAÇÃO

```
ExtractFilePath( Application.ExeName )
```

## 14) INTERCEPTAR AS TECLAS DE FUNÇÃO (f1, f2, f3...)

Primeiro, coloque a propriedade *KeyPreview* do formulário como *True*. Depois, insira este código no evento *OnKeyDown* do formulário:

```
procedure TForm1.FormKeyDown(Sender: TObject;
var Key: Word; Shift: TShiftState);
begin
if Key = VK_F5 then
showMessage('I pressed the F5 key');
end;
```

Você também pode usar as variáveis VK_F1 até VK_F12 referentes às outras teclas de função.

## 15) TRADUZINDO A MENSAGEM: "Delete Record?"

Quando clicamos sobre o botão de deleção no DBNavigator (o do sinal de menos) surge um box com a mensagem "Delete Record?" com botões Ok e Cancel. Para que apareça a mensagem em português, você deverá selecionar o componente *Table* e mudar a propriedade *ConfirmDelete* para *False* e no evento da tabela *BeforeDelete* colocar o seguinte (*flaviojr@cyber.com.br*):

```
procedure
TForm1.Table1BeforeDelete(DataSet:TDataSet);
begin
    if MessageDlg('Eliminar o
Registro?',mtConfirmation,[mbYes,mbNo],0)<>mrYes
then Abort;
end;
```

## 16) INCLUIR UM PREVIEW PERSONALIZADO NO QUICK REPORT

No relatório, criar a procedure SHOWPREVIEW, contendo:

```
Procedure Showpreview;
begin
    preview.showmodal;
end;
```

Onde preview é o nome do form criado para preview.

Não esquecer de incluir o nome da procedure na cláusula uses.

Após isso, deve-se incluir no evento CREATE do formulário principal ou do relatório o direcionamento do objeto Qprinter, como a seguir:

```
qprinter.onpreview:=showpreview;
```

Isso faz com que toda vez que você queira exibir um preview, o programa abra a rotina *showpreview*, abrindo o formulário criado, chamado *preview*.

## 17) EXECUTANDO PROGRAMAS EXTERNOS

Se você precisa abrir programas externos no seu aplicativo Delphi, como a calculadora do Windows, por exemplo, inclua a seguinte linha no seu programa:

```
WinExec('calc.exe', sw_show);
```

'calc.exe' é o nome do programa. Caso queira abrir um outro programa, altere este nome.

## 18) UTILIZANDO A TECLA ENTER PARA SALTAR DE CAMPO

Insira este código em um evento OnKeyPress de um controle de edição:

```
procedure TForm1.Edit1KeyPress(Sender:
TObject; var Key: Char);
begin
If Key = #13 Then
Begin
SelectNext(Sender as tWinControl, True, True
);
Key := #0;
```

```
end;
end;
```

## 19) TOCANDO UM SOM WAV SEM O MEDIA PLAYER

Acrescente MMSystem na Uses do começo na Unit

Utilize a API `SndPlaySound();`

Para interromper o som sem ele acabar de tocar utilize a API PlaySound(nil,0,0);

Ex: `SndPlaySound('c:\teste.wav', snd_ASync);`

PS: **snd_Loop** serve para repetir continuamente o som.

## 20) OBTER O DIRETÓRIO ONDE SEU PROGRAMA ESTÁ INSTALADO

Crie uma váriavel do tipo String e insira a seguinte linha no evento ou função desejada do formulário:

```
ExtractFilePath(Application.Exename);
```

Retornará o path atual do arquivo EXE do seu programa.

## 21) COMO BLOQUEAR UM ARQUIVO EM AMBIENTE DE REDE

É uma dica simples, mas muito importante!

Quando você programar visando uma rede e quiser bloquear um arquivo é só chamar o metodo *edit* da tabela que estiver usando.

EX: `Table1.edit;`

PS: Se o registro já estiver bloqueado ocorrerá um erro. Então, faça o seguinte :

```
try { para verificar o erro }
     Table1.edit;
exception on TDBEngineError do { o erro..}
     MessageDlg('Registro ja esta sendo
usado...!', mtInformation, [ mbOk ], 0 );
end;
```

## 22) USANDO ENTER PARA MUDAR DE CAMPO DE UM DBGRID

```
If ( Chr(Key) <> #13) Then Exit;
If ( DBGrid1.SelectedIndex + 1 <>
DBGrid1.FieldCount ) Then
DBGRid1.SelectedIndex :=
DBGRid1.SelectedIndex + 1;
```

## 23) FUNÇÃO PARA OBTER O NÚMERO DO REGISTRO ATUAL

```
Function Recno(Dataset: TDataset): Longint;
var
CursorProps: CurProps;
RecordProps: RECProps;
begin
{ Return 0 if dataset is not Paradox or dBASE
}
Result := 0;
with Dataset do
begin
if State = dsInactive then
DBError(SDataSetClosed);
Check(DbiGetCursorProps(Handle,
CursorProps));
UpdateCursorPos;
try
Check(DbiGetRecord(Handle, dbiNOLOCK, nil,
@RecordProps));
case CursorProps.iSeqNums of
0: Result := RecordProps.iPhyRecNum; { dBASE
}
1: Result := RecordProps.iSeqNum; { Paradox }
end;
except
on EDBEngineError do
Result := 0;
end;
end;
end;
```

## 24) ENVIANDO UM ARQUIVO PARA A LIXEIRA

```
uses ShellAPI;
Function DeleteFileWithUndo(sFileName :
string ) : boolean;
var
fos : TSHFileOpStruct;

begin
```

```
FillChar( fos, SizeOf( fos ), 0 );
With fos do
begin
wFunc := FO_DELETE;
pFrom := PChar( sFileName );
fFlags := FOF_ALLOWUNDO
or FOF_NOCONFIRMATION
or FOF_SILENT;
end;
Result := ( 0 = ShFileOperation( fos ) );
end;
```

## 25) CARREGAR UM CURSOR ANIMADO (*.ANI)

```
const
cnCursorID1 = 1;

begin

Screen.Cursors[ cnCursorID1 ] :=
LoadCursorFromFile('c:\win95\cursors\cavalo.ani'
);
Cursor := cnCursorID1;
end;
```

PS: O arquivo CAVALO.ANI deverá existir no diretório apontado.

## 26) TRANSFERIR O CONTEÚDO DE UM MEMO PARA O MEMOFIELD

```
var
t: TTable;
begin
t := TTable.create(self);
with t do
begin
DatabaseName := 'MyAlias'; {Nome do Alias}
TableName := 'MyTbl.db';
open;
edit;
insert;
FieldByName('TheField').assign(Memo1.lines);
post; { Requerido!!!}
close;
end;
end;
```

## 27) CAPTURANDO O CONTEÚDO DO DESKTOP

Coloque o este código no evento *FormResize* do formulário.

```
procedure TForm1.FormResize(Sender: TObject);
var
R : TRect;
DC : HDc;
Canv : TCanvas;
begin
R := Rect( 0, 0, Screen.Width, Screen.Height
);
DC := GetWindowDC( GetDeskTopWindow );
Canv := TCanvas.Create;
Canv.Handle ;= DC;
Canvas.CopyRect( R, Canv, R );
ReleaseDC( GetDeskTopWindow, DC );
end;
```

## 28) ESCREVENDO UM TEXTO DIAGONAL USANDO O CANVAS

```
procedure TForm1.Button1Click(Sender: TObject);
var
begin
     with Form1.Canvas do begin
Font.Name := 'Arial';
Font.Size := 24;
tf := TFont.Create;
tf.Assign(Font);
GetObject(tf.Handle, sizeof(lf), @lf);
lf.lfEscapement := 450;
lf.lfOrientation := 450;
tf.Handle := CreateFontIndirect(lf);
Font.Assign(tf);
tf.Free;
TextOut(20, Height div 2, 'Texto Diagonal!');
     end;
 end;
```

## 29) EXTRAIR UM ICONE DE UM DETERMINADO APLICATIVO

Para extrair ícones de um executável, deve-se usar a função da API Extraction. Ela usa três parâmetros:

Instance - Instância da aplicação

FileName - Nome do executável. Deve ser um PChar

NumIcon - Número do ícone a ser recuperado. Se for Word(-1), a função retorna a quantidade de ícones do executável.

Coloque ShellAPI em uses no começo da unit.

```
procedure TForm1.Button1Click(Sender:
TObject);
var
```

```
IconIndex : word;
h : hIcon;
begin
IconIndex := 0;
h :=    ExtractAssociatedIcon(hInstance,'C:
\WINDOWS\NOTEPAD.EXE',IconINdex);
DrawIcon(Form1.Canvas.Handle,10,10,h);
end;
```

## 30) ALINHANDO ITEMS DO MENU À DIREITA

Para alinhar itens do menu principal à direita, deve-se utilizar o seguinte código:

```
{Isto justifica todos itens à direita do
selecionado}

procedure SetJustify(Menu: TMenu; MenuItem:
TMenuItem; Justify: Byte);
{$IFDEF WIN32}
var
ItemInfo: TMenuItemInfo;
Buffer: array[0..80] of Char;
{$ENDIF}
begin
{$IFDEF VER80}
  MenuItem.Caption := Chr(8) +
MenuItem.Caption;
{$ELSE}
  ItemInfo.cbSize := SizeOf(TMenuItemInfo);
  ItemInfo.fMask := MIIM_TYPE;
  ItemInfo.dwTypeData := Buffer;
  ItemInfo.cch := SizeOf(Buffer);
  GetMenuItemInfo(Menu.Handle,
MenuItem.Command, False, ItemInfo);
  if Justify = 1 then
    ItemInfo.fType := ItemInfo.fType or
MFT_RIGHTJUSTIFY;
    SetMenuItemInfo(Menu.Handle,
MenuItem.Command, False, ItemInfo);
{$ENDIF}
end;
```

## 31) ABRIR AUTOMATICAMENTE SEU NAVEGADOR PADRÃO E CARREGAR A PÁGINA DETERMINADA PELO LINK

1º Declare o procedure na seção *Public* da unit.

procedure JumpTo(const aAdress: String);

2º Coloque a cláusula *ShellAPI* na uses no início da unit.

```
procedure TForm1.JumpTo(const aAdress:
String);
var
     buffer: String;
begin
     buffer := 'http://' + aAdress;
     ShellExecute(Application.Handle, nil,
PChar(buffer), nil, nil, SW_SHOWNORMAL);
end;

procedure TForm1.Label1Click(Sender:
TObject);
begin
     JumpTo('www.geocities.com/
SiliconValley/Way/1497');
end;
```

## 32) COPIAR REGISTROS DE UMA TABELA PARA OUTRA INCLUINDO VALORES NULL

```
procedure TtableCopiaRegistro(Origem, Desti-
no: Ttable);
begin
     with TabelaOrig do
      begin
          {Inicia um contador para os campos
da TabelaOrig}
          for i := 0 to FieldCount -1 do
          {Este if verifica se o campo da
TabelaOrig é NULL, se for, atribui seu valor
ao
             campo da TabelaDest}
           if not Fields[i].IsNull then
TabelaDest.Fields[i].Assign(Fields[i]);
     end; {end with}
end;
```

Este exemplo funcionará com todos os tipos de campos se você tiver acabado de criar a TabelaDest.
Para criar um dado valor NULL :
Fields[i].Clear

## 33) DELETAR ARQUIVOS DE UM DIRETÓRIO COM O CARACTERE CURINGA '*'

```
    procedure
TForm1.SpeedButton1.Click(Sender: TObject);
     var
      SearchRec: TSearchRec;
      Result: Integer;
     begin
```

```
     Result:=FindFirst('c:\teste\*.*',
faAnyFile, SearchRec);
     while result=0 do
          begin

DeleteFile('c:\teste\'+SearchRec.Name);
               Result:=FindNext(SearchRec);
          end;
     end;
```

## 34) CAPTURAR A LISTA DE ALIASES DISPONÍVEIS

Tudo o que você precisa é de um componente TSession, um componente **TListBox e uma String List.**

Defina a propriedade *SessionName* do TSession para *'Session'*. Utilize o seguinte código:

```
procedure TForm1.Button3Click(Sender:
TObject);
var
MyStringList: TStringList;
          i: integer;
begin
          MyStringList :=
TStringList.Create;
        Session.GetAliasNames(MyStringList);
          for I := 0 to MyStringList.Count
- 1 do
     ListBox1.Items.Add(MyStringList[I]);
end;
```

Utilize o Help do TSession e consulte seus métodos para ver, por exemplo, como capturar o diretório ou caminho de um Alias com o método 'GetAliasParams'.

## 35) ABRIR E FECHAR A BANDEJA DO DRIVE DE CD-ROM

```
{Para Abrir:}
mciSendString('Set cdaudio door open wait',
nil, 0, handle);

{Para Fechar:}
mciSendString('Set cdaudio door closed wait',
nil, 0, handle);
```

## 36) UTILIZANDO O CODE EXPLORER

A versão 4 do Borland Delphi está recheada de novos recursos em várias áreas do produto (IDE, Internet, linguagem, aplicações multi-tier, entre outras).

Uma das primeiras coisas que notamos quando abrimos o editor de código do Delphi 4 é a presença de um painel ancorado na lateral esquerda, contendo todos os tipos, classes, propriedades, métodos, variáveis globais, rotinas globais e interfaces contidos na unit selecionada. Este painel é chamado Code Explorer e seu objetivo é tornar mais fácil a navegação entre as units do projeto e automatizar alguns processos envolvidos na criação de classes.

O Code Explorer também permite que você navegue diretamente para as declarações que são apresentadas nele, bastando dar um duplo clique com o mouse sobre a declaração desejada. Novas declarações podem ser feitas também usando o Code Explorer, facilitando o desenvolvimento do código. Por exemplo, vamos supor que você queira criar uma nova função chamada *Calculo* com dois parâmetros do tipo real e que retornará também um real. Para isso, deve-se seguir os seguintes passos:

Selecione a pasta *Variable/Constants* dentro do *Code Explorer*. Dê um clique com o botão direito do mouse e selecione *New* no menu que aparecerá, como mostra a figura a seguir.

O *Code Explorer* apresentará um novo item que permite identificar qual o tipo de declaração que está sendo feita, por meio da informação que o programador passar a ele. No nosso exemplo, digitaremos o cabeçalho da função Calculo como segue :

```
function Calculo(x, y : real) : real;
```

Após digitar a declaração e pressionar *<Enter>*, o Code Explorer criará automaticamente o cabeçalho da função na seção *Interface* e a sua implementação na seção *Implementation da unit*, como mostra a figura seguinte, evitando assim que ocorram erros de declaração no interior do código.

Um outro recurso que o Code Explorer fornece é conhecido como *Class Completion*. O programador pode, dentre outras facilidades, criar apenas o básico da declaração de uma propriedade e, com o simples toque de um atalho no teclado, o Code Explorer completará a declaração. Como exemplo, vamos declarar uma propriedade chamada *Cor* do tipo *TColor* dentro da nossa classe *TForm 1*.

```
type
```

```
      TForm1=class(TForm)
private
            {declarações privadas}
      public
            {declarações públicas}
property Cor:TColor;
  end;
```

Agora, com o cursor posicionado sobre a declaração da propriedade, pressionamos *<Ctrl><Shift>C* e o Code Explorer completará toda a declaração da estrutura da classe.

## 37) COPIANDO ARQUIVOS VIA DELPHI

```
Function
CopiaArquivo(scrname,destname:string):byte;
var
      source,destination:file;
      buffer:array[1..1024] of byte;
      readcnt,writecnt:word;
      pname,dname,fname,ename:String;
{USO:
R:=COPIAARQUIVO('C:\diretorio
\FILE.EXT','C:\diretorio\FILE.EXT');
Devolve 0=Ok, 1=Erro no Origem, 2=Erro no
Destino, 3=Disco Cheio}
begin
      AssignFile(source,scrname);
      Try
            Reset(source,1);
      Except
            CopiaArquivo:=1;
            Exit;
      end;

      If destname[length(destname)]='\' then
      begin
            pname:=scrname;

destname:=destname+separa(scrname,'\',Ocorre(scrname,'\')+1);
      end;
      AssignFile(destination,destname);
      Try
            Rewrite(destination,1);
      Except
            CopiaArquivo:=2;
            Exit;
      end;

      Repeat

BlockRead(source,buffer,sizeof(buffer),readcnt);
            Try

BlockWrite(destination,buffer,readcnt,writecnt);
            Except
                  CopiaArquivo:=3; {Disco
Cheio?}
                  Exit;
            end;
      until (readcnt=0) or
(writecnt<>readcnt);
      CloseFile(destination);
      CloseFile(source);
      CopiaArquivo:=0;
end;
```

## 38) ABRIR UM TCOMBOBOX SEM CLICÁ-LO

```
ComboBox1.DroppedDown := True;
```

## 39) MUDAR A COR DA CÉLULA ATIVA DO DBGRID

A rotina abaixo deverá ser colocada no evento OnDrawDataCell, do DBGrid.

```
procedure TForm1.DBGrid1DrawDataCell(Sender:
TObject; const Rect: TRect; Field: TField;
State: TGridDrawState);
begin
      if gdFocused in State then
            with (Sender as TDBGrid).Canvas do
            begin
                  Brush.Color:=clRed;
                  FillRect(Rect);
            TextOut(Rect.Left, Rect.Top,
Field.AsString);
            end;
end;
```

## 40) COMO INCREMENTAR UM MÊS NUMA DATA

```
IncMonth(Data, 1);
```

No exemplo, a variável Data é do tipo TDateTime.

Aprenda a Programar

Em respeito ao jornaleiro a Digerati não trabalha com assinaturas

**Atendimento ao leitor**
Fone:(11)3217-2626(9h às 21h) – suporte@digerati.com.br
Marcos Raul de Oliveira, Eduardo Rodrigues, Rodrigo França e Thiago Sobrinho
**Atendimento de vendas**
Fone: (11) 3217-2600 – vendas@digerati.com.br
Luana Aguiar e Ana Paula Venancio

## Revista Aprenda a Programar

**Editor**
Marcelo Barbão *(mbarbao@digerati.com.br)*
**Editor assistente**
Maurício Martins *(mauricio@digerati.com.br)*
**Redatores**
Bruno Cesar, João Marinho e Fernando Wiek
**Arte**
Helber Bimbo, Marina Fiorese e Fábio Augusto
**Colaboraram nesta edição**
Marcelo Jaloto Machado, Melovis
**Revisão**
Priscila Cassettari, Cíntia Yamashiro
**Departamento Multimídia**
Design e Programação: Rodrigo Rudiger
Seleção de Programas: Juliano Barreto e João Henrique
Vídeo: Felipe Madureira
**Departamento de Internet**
Tarcila Broder, Carlos Sivalli Ignatti

Os artigos assinados não refletem necessariamente a opinião da revista, e sim de seus autores.

Mais uma publicação da DIGERATI
a editora especialista em comunidade digital

**Digerati Comunicação e Tecnologia Ltda**
Rua Haddock Lobo, 347 – 12°. Andar
CEP 01414-001 São Paulo SP
Fone: (11) 3217-2600 Fax: (11) 3217-2617
www.digerati.com

**Diretores**
Alessandro Gerardi – (gerardi@digerati.com.br)
Luis Afonso G. Neira – (afonso@digerati.com.br)
Alessio Fon Melozo – (alessio@digerati.com.br)
**Diretor Comercial**
René Luiz Cassettari – (rene@digerati.com.br)
**Representante Comercial no E.U.A.**
Multimedia, Inc - Tel. + 1-407-903-5000 Ext.222 Fax +1-407-363-9809
Fernando Mariano – (info@multimediausa.com)
**Marketing**
Érica V. Cunha, Simone Siman, Carlos Ignatti, José Antonio Martins
**Assessoria de imprensa**
Simone Siman – (siman@digerati.com.br)
**Recursos Humanos**
Viviane Cardoso – (viviane@digerati.com.br)
**Logística de Produção**
Pierre Abreu – (pierre@digerati.com.br)
**Tecnologia da Informação**
Flavio Tâmega – (flavio@digerati.com.br)

**Impressão e Acabamento**
Oceano Indústria Gráfica Ltda.
Fone: (11) 4446-6544
**Distribuidor Exclusivo para bancas de todo o Brasil**
Fernando Chinaglia Distribuidora SA
Fone: (21) 3879-7766

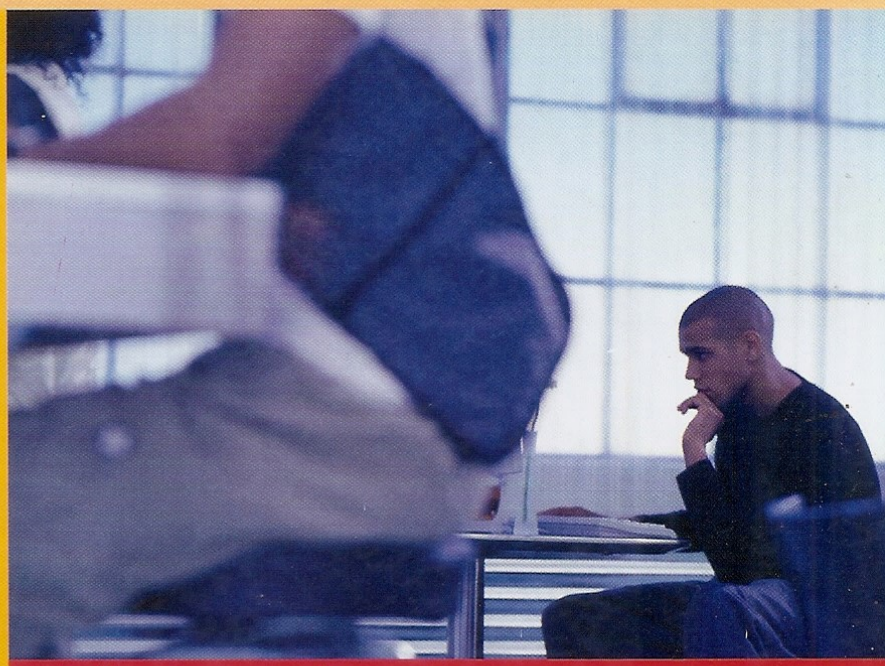
ANER www.aner.org.br IVC

Digerati é a editora especializada em comunidade Digital

Digerati Home, a informática do seu dia-a-dia

# educação

*sf* 1Desenvolvimento das faculdades do ser humano. 2Desenvolvimento e aperfeiçoamento de uma função pelo próprio exercício. 3Ensino. 4Civilidade.



# E-Learning

1Educação digital. 2Professor digital. 3Tecnologia a serviço do ensino e do aprendizado.


**Revista E-Learning**
Revista mais CD-ROM por apenas R$ 11,90.
Nas bancas ou no site www.digerati.com

Complete a sua coleção com comodidade, compre na loja virtual da Digerati

www.digerati.com