

Capítulo 2

Olá, Windows 95

Se você é novo em programação para um ambiente gráfico tal como o Microsoft Windows 95, provavelmente o considerará muito diferente de tudo o que conhece. O Windows tem a reputação de ser fácil para os usuários, mas difícil para os programadores. É muito comum os novatos sentirem-se confusos com a arquitetura do Windows e com a estrutura dos aplicativos que rodam sob o sistema. Se isso acontecer com você, não pense que está perdendo alguma parte vital do seu cérebro necessária para tornar-se um programador Windows bem-sucedido. A confusão inicial é normal, e não permita que outras pessoas digam o contrário.

A programação baseada no Windows é estranha e confusa. Ela com certeza não é imediatamente óbvia, e pode levar um certo tempo até que você experimente a emoção de exclamar "Eureka!" (também conhecida como a revelação "Agora sim, entendi!", que os professores gostam tanto de ouvir de seus alunos). Uma estimativa comum é que os programadores precisam passar por uma curva de aprendizado de 6 meses antes de estarem capacitados a escrever programas Windows e, mesmo depois disso, o aprendizado não termina. Minha única esperança é que este livro venha a cortar algumas semanas (ou talvez um ou dois meses) da curva de aprendizagem típica.

Você pode provavelmente perguntar: "Se a programação para o Windows é tão difícil, por que se ocupar com ela?".

Bem, a resposta fácil é: "Provavelmente você não tem muita escolha". Afinal, o Windows alcançou uma penetração tão generalizada no mercado dos microcomputadores compatíveis com o PC que a programação para o MS-DOS comum (usando o modo de caracteres ou o modo gráfico) não é mais viável. Se você desenvolve aplicativos comerciais, os comentaristas de software praticamente ignorarão seu produto se ele não for voltado para o Windows. Se você desenvolve programas especializados em uma grande empresa, seus

usuários (e provavelmente seus empregadores) não apreciarão o fato de seu programa não se integrar perfeitamente aos aplicativos Windows existentes que eles utilizam.

Porém, existem melhores razões para escolher o Windows como sua plataforma.

A DIFERENÇA DO WINDOWS

O Windows oferece consideráveis vantagens para o usuário e para o programador em relação ao ambiente convencional do MS-DOS. Os benefícios para ambos são na verdade bem parecidos, pois o trabalho de um programador é dar ao usuário o que ele precisa e quer. O Windows torna isso possível.

A Interface Gráfica com o Usuário

O Windows é uma interface gráfica com o usuário (GUI, de Graphical User Interface), algumas vezes também chamada "interface visual" ou "ambiente gráfico com janelas". Os conceitos que criaram esse tipo de interface originaram-se em meados dos anos 70, com o trabalho pioneiro desenvolvido no Centro de Pesquisa da Xerox (PARC), em Palo Alto, para máquinas, como o Alto e o Star e ambientes como o Smalltalk. O trabalho foi posteriormente aplicado e popularizado pela Apple Computer, inicialmente no fracassado computador Lisa e um ano mais tarde no bem-sucedido Macintosh, lançado em janeiro de 1984.

Desde o lançamento do Macintosh, novas interfaces gráficas com o usuário floresceram na indústria dos computadores pessoais e não-pessoais. É evidente que a interface gráfica com o usuário é agora (nas palavras de Charles Simonyi, da Microsoft) o mais importante e único "grande consenso" na indústria de computadores pessoais.

Princípios e Conceitos da GUI

Todas as interfaces gráficas com o usuário utilizam gráficos em monitores formados por mapas de bits. Os gráficos fazem melhor utilização da tela, transmitindo informações de uma maneira visual mais rica, e possibilitam o que se chama de WYSIWYG (what you see is what you get, o que você vê é aquilo que obtém) das figuras e do texto formatado para o documento que será impresso.

Antigamente, o monitor de vídeo era utilizado somente para reproduzir o texto que o usuário digitava no teclado. Em uma interface gráfica com o usuário, o monitor na verdade se transforma em uma fonte de entrada de dados do usuário. A tela do vídeo agora mostra vários objetos gráficos na forma de ícones e dispositivos de entrada, como botões e barras de rolagem. Usando o teclado (ou mais diretamente, um dispositivo apontador, como um mouse), o usuário pode manipular diretamente esses objetos na tela. Os objetos gráficos podem ser arrastados, os botões podem ser pressionados e as barras de rolagem podem ser roladas.

A interação entre o usuário e o programa torna-se assim mais íntima. Em vez da via de mão única do teclado para o programa e para a tela do vídeo, o usuário passa a interagir diretamente com os objetos na tela.

Uma Interface Consistente

Os usuários não querem mais gastar um longo período de tempo aprendendo a usar o computador ou a dominar um novo programa. O Windows ajuda porque todos os programas Windows têm a mesma aparência e modo de operação fundamental. O programa ocupa uma janela – uma área retangular da tela. Ele é identificado por uma barra de título. A maioria das funções do programa é iniciada por meio do menu do programa. A exibição de informações grandes demais para caber em uma única tela pode ser vista usando-se as barras de rolagem. Alguns itens de menu invocam caixas de diálogo, nais quais o usuário insere informações adicionais. Uma caixa de diálogo encontrada em praticamente todos os grandes programas Windows abre um arquivo. Essa caixa de diálogo tem aparência igual (ou muito similar) em diferentes programas Windows, e é quase sempre acionada com a mesma opção de menu.

Uma vez que você saiba como operar um programa Windows, está com meio caminho andado para aprender facilmente qualquer outro. Os menus e as caixas de diálogo permitem ao usuário testar o novo programa e explorar seus recursos. A maioria dos programas Windows pode ser controlada por meio do teclado e pelo mouse. Embora a maioria das funções possa ser executada por meio do teclado, o uso do mouse geralmente facilita as tarefas.

Na perspectiva do programador, uma interface com o usuário consistente possibilita o uso de rotinas internas do Windows para a criação de menus e caixas de diálogo. Todos os menus têm a mesma interface de mouse e teclado porque o próprio Windows, e não o programa aplicativo, cuida desta tarefa.

A Vantagem da Multitarefa

Embora algumas pessoas continuem a questionar se a multitarefa é realmente necessária em um computador monousuário, os usuários estão definitivamente prontos para utilizá-la e podem beneficiar-se dela. A popularidade de programas residentes em RAM para o MS-DOS, como o Sidekick, prova isso. Embora programas residentes não sejam, rigorosamente falando, programas multitarefa, eles permitem uma rápida mudança de contexto, e isto envolve muito dos conceitos de multitarefa.

Sob o Windows, todo programa torna-se na verdade residente em RAM. Muitos programas Windows podem ser mostrados e executados ao mesmo tempo. Cada um deles ocupa uma janela retangular na tela. O usuário pode mover as janelas na tela, alterar o tamanho delas, alternar e transferir dados de um programa para outro. Como a tela se parece um pouco com uma mesa de trabalho (antes de esta ser dominada pelo computador, é claro), diz-se que o Windows usa a metáfora de uma mesa de trabalho para mostrar múltiplos documentos.

As versões anteriores do Windows usavam um sistema de multitarefa denominado “não-preemptivo”. Isso significava que o Windows não usava o temporizador do sistema para alocar tempo de processamento entre os vários programas em execução sob o sistema. Os próprios programas tinham de voluntariamente ceder o controle para que outros programas pudessem ser executados. Sob o Windows 95, a multitarefa é preemptiva, e os programas podem dividir-se em múltiplas cadeias (threads) de execução, que parecem rodar concorrentemente.

Gerenciamento de Memória

Um sistema operacional não pode implementar a multitarefa sem fazer algo com o gerenciamento de memória. À medida que novos programas são carregados e os mais antigos se encerram, a memória fica fragmentada. O sistema precisa agrupar o espaço livre na memória. Para isso, ele precisa mover blocos de código e de dados.

Mesmo o Windows 1, rodando em um microprocessador 8088, podia executar esse tipo de gerenciamento de memória. No modo real, isso foi um feito surpreendente de engenharia de software. Os programas que rodam sob o Windows podem exigir mais memória do que disponível; eles podem conter mais código e dados do que cabe na memória ao mesmo tempo. O Windows pode descarregar parte do código do programa da memória e mais tarde recarregá-lo no arquivo do programa .EXE. O usuário pode executar várias cópias (ou ocorrências) de um mesmo programa. Essas ocorrências compartilham o mesmo código na memória. Os programas em execução no Windows podem compartilhar rotinas localizadas em outros arquivos .EXE chamados “bibliotecas de ligações dinâmicas”. O Windows contém um mecanismo para ligar o programa com as rotinas nessas bibliotecas durante a execução. O próprio Windows é um conjunto de bibliotecas de ligações dinâmicas.

Assim, mesmo no Windows 1, o limite de 640 KB de memória da arquitetura do PC foi efetivamente ampliado, sem requerer memória adicional. Porém, a Microsoft não parou por aí. O Windows 2 deu aos aplicativos acesso à memória expandida (EMS), e o Windows 3 rodava no modo protegido, dando aos aplicativos acesso a 16 MB de memória estendida. E agora o Windows 95 afasta essas velhas restrições, pois é um sistema operacional de 32 bits com espaço de endereçamento de memória linear.

A Interface Gráfica Independente do Dispositivo

O Windows é uma interface gráfica, e os programas Windows podem utilizar gráficos e texto formatado tanto no vídeo como na impressora. Uma interface gráfica não é somente mais atraente em aparência, mas também pode transmitir grande número de informações ao usuário.

Os programas feitos para o Windows não acessam diretamente o hardware dos dispositivos gráficos como a impressora e o monitor de vídeo. O Windows contém uma linguagem de programação de gráficos (chamada de Graphics Device Interface, ou GDI) que permite a fácil apresentação de gráficos e texto formatado. Um programa criado para o ambiente Windows funcionará com qualquer placa de vídeo e qualquer impressora para a qual exista um controlador de dispositivo disponível para o Windows. O programa nem precisa determinar que tipo de dispositivo está conectado ao sistema.

Incluir uma interface gráfica independente do dispositivo no IBM-PC não foi uma tarefa fácil para os criadores do Windows. O projeto do PC baseou-se no princípio da arquitetura aberta. Os fabricantes de hardware foram encorajados a desenvolver periféricos para o PC e o fizeram em grande número. Embora diversos padrões tenham emergido, os programas convencionais para o MS-DOS precisam suportar individualmente diferentes configurações de hardware. Por exemplo, é bastante comum que um programa de processamento de texto

para o MS-DOS seja vendido com um ou dois disquetes contendo pequenos arquivos, um para cada tipo de impressora. Os programas para Windows 95 não requerem esses controladores, pois o suporte é responsabilidade do próprio Windows.

Comprometimento com o Windows

A programação para o Windows é uma escolha do tipo tudo-ou-nada. Por exemplo, você não pode criar um aplicativo MS-DOS e usar o Windows somente nos gráficos. Se quiser usar o Windows, precisa comprometer-se a criar um aplicativo inteiramente voltado para o Windows.

A razão para isso ficará mais evidente quando você estudar a estrutura de um programa Windows. No Windows, tudo está interconectado. Se você quiser desenhar um gráfico na tela do vídeo, precisará de algo chamado "handle (ou indicativo) do contexto do dispositivo". Para obtê-lo, precisará do "handle da janela". Para conseguir este último, precisará criar uma janela e estar preparado para receber "mensagens" para ela. Para receber e processar as mensagens, você precisará de um "procedimento de janela". E aí você estará codificando um programa Windows. Somente os que decidem abandonar o chão é que podem voar.

Chamadas às Funções

O Windows 95 suporta mais de 1.000 funções que os aplicativos podem usar. É altamente improvável que você consiga memorizar a sintaxe de todas elas. A maioria dos programadores gasta boa parte de seu tempo consultando as chamadas às várias funções, na forma impressa ou em referências on-line.

Toda função do Windows tem um nome descritivo com uma mistura de letras maiúsculas e minúsculas, como, por exemplo, *CreateWindow*. Essa função (como você pode deduzir) cria uma janela para o seu programa. Outro exemplo: a função *IsClipboardFormatAvailable* determina se a Área de Transferência contém dados em determinado formato.

Todas as funções do Windows estão declaradas em um arquivo de cabeçalho chamado WINDOWS.H, e este arquivo de cabeçalho inclui muitos outros arquivos. Os arquivos de cabeçalho são fornecidos em qualquer ambiente de programação baseado em C que suporte o Windows 95. Eles são uma parte importante da documentação técnica do Windows. Imprima uma cópia desses arquivos ou use um editor de programas para poder consultá-los com rapidez.

No seu programa Windows, você pode usar as chamadas às funções do Windows da mesma forma que usa as funções da biblioteca do C, como *strlen*. A principal diferença é que o código para as funções da biblioteca C é ligado ao código do seu programa, enquanto o código para as funções do Windows está localizado fora do seu programa, nas bibliotecas de ligação dinâmica (DLLs).

Quando você roda um programa Windows, ele se comunica com o Windows por meio de um processo chamado "ligação dinâmica". Um arquivo .EXE do Windows contém referências às várias bibliotecas de ligações dinâmicas que ele utiliza e às funções ali contidas. A maioria dessas DLLs está localizada no subdiretório SYSTEM do seu diretório Windows.

Quando um programa Windows é carregado na memória, as chamadas no programa são resolvidas para apontar para as entradas das funções nas bibliotecas de ligação dinâmica, que também são carregadas na memória, caso ainda não estejam ali.

Quando você liga um programa Windows para produzir um executável, precisa ligá-lo com “bibliotecas de importação” especiais fornecidas pelo seu ambiente de programação. Essas bibliotecas de importação contêm os nomes das bibliotecas de ligação dinâmica e as informações de referência para todas as funções do Windows. O linker usa essas informações para construir a tabela no arquivo .EXE que o Windows utilizará para acertar as chamadas às funções do Windows ao carregar o programa.

Programação Orientada a Objeto

Ao programar para o Windows, você estará realmente engajado em um tipo de programação orientada a objeto. Isso é principalmente evidente no objeto com o qual você mais trabalhará no Windows – o objeto que dá ao Windows o seu nome, o objeto que você logo verá tomar características antropomórficas, o objeto que surgirá nos seus sonhos, o objeto conhecido como “janela”.

Como mencionei anteriormente, as janelas são áreas retangulares na tela. Uma janela recebe entrada do usuário a partir do teclado ou do mouse e exibe uma saída gráfica.

Uma janela de aplicativo normalmente contém a barra de título do programa, menus, molduras de tamanho e possivelmente uma ou duas barras de rolagem. As caixas de diálogo são janelas adicionais. Além disso, a superfície da caixa de diálogo sempre contém diversas janelas adicionais chamadas “janelas-filhas”, que têm a forma de botões, caixas de verificação, campos para a inserção de texto, caixas de listagem e barras de rolagem.

O usuário vê essas janelas como objetos na tela e interage diretamente com eles pressionando um botão ou rolando a barra de rolagem. Curiosamente, a perspectiva do programador é análoga à do usuário. A janela recebe essa entrada do usuário na forma de “mensagens”. Uma janela também usa mensagens para se comunicar com as outras janelas.

Compreender essas mensagens é um dos obstáculos que você terá de superar para tornar-se um programador no ambiente Windows.

Arquitetura Orientada por Mensagens

Na primeira vez em que vi uma interface gráfica com o usuário, fiquei confuso. A demonstração incluía um processador de texto rudimentar rodando em uma janela. O processador reformatava o texto quando o tamanho da janela era alterado.

Era óbvio para mim que o sistema operacional estava cuidando dos detalhes da lógica de alteração do tamanho da janela e que o programa podia responder a esta função do sistema. Como o programa *sabia* quando sua janela havia sido alterada? Que mecanismo do sistema operacional era usado para transmitir essa informação à janela? Minha experiência anterior com programação de nada servia para compreender como isso funcionava.

Isso mostra que a resposta a essa questão depende da compreensão da arquitetura usada nas interfaces gráficas do usuário. No Windows, quando o usuário altera o tamanho de uma janela, o Windows envia uma mensagem ao programa informando o novo tamanho da janela. O programa pode então ajustar o conteúdo da sua janela para refletir o novo tamanho.

"O Windows envia uma mensagem ao programa." Suponho que você não tenha lido essa frase sem alguma estranheza. Que diabos isso significa? Estamos falando sobre código de programa aqui, não sobre um sistema de correio eletrônico. Como pode um sistema operacional enviar mensagens a um programa?

Quando digo que "o Windows envia uma mensagem ao programa", quero dizer que o Windows chama uma função dentro do programa. Os parâmetros dessa função descrevem a mensagem particular. Essa função no seu programa Windows é conhecida como "procedimento de janela".

O Procedimento de Janela

Você sem dúvida está acostumado com a idéia de um programa chamar o sistema operacional. É assim que um programa abre um arquivo no disco, por exemplo. Você pode não estar acostumado é com a idéia de um sistema operacional fazer chamadas a um programa. Porém, isso é fundamental na arquitetura orientada a objeto do Windows 95.

Toda janela que um programa cria tem um procedimento de janela associado. Esse procedimento é uma função que poderia estar no próprio programa ou na biblioteca de ligações dinâmicas. O Windows envia uma mensagem para uma janela chamando o procedimento de janela. O procedimento de janela executa uma tarefa baseando-se na mensagem e em seguida retorna o controle ao Windows.

Mais precisamente, uma janela é sempre criada com base em uma "classe de janela". A classe de janela identifica o procedimento que processa as mensagens para a janela. O uso de uma classe de janela permite que múltiplas janelas sejam baseadas na mesma classe de janela e, portanto, usem o mesmo procedimento. Por exemplo, todos os botões em todos os programas Windows estão baseados na mesma classe de janela. Essa classe está associada a um procedimento (localizado na biblioteca de ligações dinâmicas USER.EXE) que processa as mensagens para todas as janelas com botões.

Na programação orientada a objeto, um "objeto" é uma combinação de código e dados. Uma janela é um objeto. O código é o procedimento de janela. Os dados são informações retidas pelo procedimento de janela e pelo Windows para cada janela e classe de janela que exista no sistema.

Um procedimento de janela processa as mensagens à janela. Na maioria das vezes, essas mensagens informam a janela sobre a entrada de dados pelo teclado ou pelo mouse. É assim que uma janela que contém um botão de seleção sabe que este foi pressionado. Outras mensagens informam a janela quando ela teve seu tamanho alterado ou quando ela precisa ser redesenhada.

No início da execução de um programa, o Windows cria uma "fila de mensagens" para o programa. Essa fila de mensagens armazena as mensagens para todas as janelas que o

programa possa criar. O programa inclui um pequeno bloco de código chamado “laço de mensagens” para retirar essas mensagens da fila e despachá-las ao procedimento de janela apropriado. Outras mensagens são enviadas diretamente ao procedimento de janela, sem passar primeiro pela fila.

Se os seus olhos estão começando a arder com essa descrição excessivamente abstrata da arquitetura do Windows, talvez seja útil ver como a janela, a classe de janela, o procedimento de janela, a fila de mensagens, o laço de mensagens e as mensagens à janela combinam-se no contexto de um programa de verdade.

SEU PRIMEIRO PROGRAMA WINDOWS

No clássico livro *The C Programming Language* (2 ed., Prentice Hall, 1988), Brian Kernighan e Dennis Ritchie começam a discutir a linguagem C com o agora famoso programa “Olá, pessoal”:

```
#include <stdio.h>
main ()
{
    printf ("Olá, pessoal\n");
}
```

No restante deste capítulo, mostrarei um programa análogo criado para o Microsoft Windows. Ele se chama OLAWIN, e cria uma janela contendo a mensagem “Olá, Windows 95!” e também reproduz um arquivo de som com minha voz recitando esta frase.¹

Antes que você morra de susto ao ver o código para o OLAWIN, deixe-me avisá-lo agora de que o arquivo-fonte para o programa tem mais de 80 linhas. A maioria dessas 80 linhas é formalidade de inicialização que todo programa Windows deve seguir.

Em vez de perguntar por que o programa “Olá, Windows 95!” é tão longo, vamos perguntar por que o tradicional “Olá, pessoal” é tão pequeno e simples.

O que Há de Errado com Este Programa?

O modelo de saída para o programa “Olá, pessoal” e outros programas tradicionais em C é uma antiga máquina conhecida como teletipo. O teletipo é semelhante a uma máquina de escrever com um rolo de papel contínuo. Em um passado não muito distante, os programadores sentavam-se diante do teletipo e digitavam os comandos que eram impressos no papel.

A metáfora do teletipo foi estendida ao monitor de vídeo nos primórdios dos computadores. O monitor de vídeo tornou-se um “teletipo com tela” que simplesmente rolava quando o texto chegava à última linha.

1. N. R. T: Este arquivo de som foi mantido no original em inglês para que o leitor possa ouvir a voz do próprio autor.

Como o tradicional programa “Olá, pessoal” pode imprimir sem dizer ao sistema operacional em que dispositivo de saída o texto deve aparecer? A razão é que existe apenas um dispositivo de saída – o monitor de vídeo usado como se fosse um teletipo. Se o usuário quiser que a impressão seja feita em outro meio, precisará redirecioná-la na linha de comando.

Como o programa pode mostrar texto sem dizer ao sistema operacional em que posição do dispositivo de saída o texto deve aparecer? Como o texto sempre aparece na posição do cursor, provavelmente ele aparecerá na próxima linha após a carga do programa. Suponha que você queira apresentar a mensagem “Olá, pessoal” no centro da tela. Você teria de usar alguns códigos de controle dependentes do dispositivo para primeiro mover o cursor para a posição desejada.

Suponha que você quisesse executar ao mesmo tempo vários programas para imprimir a mensagem “Olá, pessoal” e ver o efeito deles na tela. Que bagunça! As ocorrências (cópias) do programa interfeririam umas nas outras. Não há nada na metáfora dos teletipos para separar as impressões de diversos programas em execução concorrente.

É também interessante que você veja a mensagem “Olá, pessoal” mesmo após o término do programa. Em vez de limpar todos os vestígios de sua existência, o programa deixa a mensagem na tela.

O programa “Olá, pessoal” é simples porque é destinado a uma época mais simples, computadores mais simples e dispositivos de saída também mais simples. Ele não está na mesma categoria do que chamamos software moderno, nem está jogando o mesmo jogo.

Os Arquivos do OLAWIN

Dois dos três arquivos necessários para criar o programa OLAWIN são mostrados na Figura 2.1. São eles o arquivo OLAWIN.MAK para o utilitário Make e o OLAWIN.C, que é o código-fonte. O terceiro arquivo está armazenado no disquete que acompanha o livro como OLAWIN.WAV; este é um arquivo de forma de onda contendo o texto por mim pronunciado.

OLAWIN.MAK

```
#-----
# OLAWIN.MAK - Arquivo para o NMAKE
#-----
```

```
olawin.exe : olawin.obj
$(LINKER) $(GUIFLAGS) -OUT:olawin.exe olawin.obj $(GUILIBS)

olawin.obj : olawin.c
$(CC) $(CFLAGS) olawin.c
```

Figura 2.1 O programa OLAWIN.

OLAWIN.C

```
-----  
OLAWIN.C - Exibe "Olá, Windows 95!" na área do cliente  
(c) Charles Petzold, 1996  
-----*/  
  
#include <windows.h>  
  
LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);  
  
int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt,  
                     PSTR szLinhaCmd, int iCmdMostrar)  
{  
    static char szNomeAplic[] = "Olawin";  
    HWND      hjan;  
    MSG       msg;  
    WNDCLASSEX classejan;  
  
    classejan.cbSize          = sizeof (classejan);  
    classejan.style           = CS_HREDRAW | CS_VREDRAW;  
    classejan.lpfnWndProc     = ProcJan;  
    classejan.cbClsExtra      = 0;  
    classejan.cbWndExtra      = 0;  
    classejan.hInstance        = hCopia;  
    classejan.hIcon           = LoadIcon (NULL, IDI_APPLICATION);  
    classejan.hCursor          = LoadCursor (NULL, IDC_ARROW);  
    classejan.hbrBackground    = (HBRUSH) GetStockObject (WHITE_BRUSH);  
    classejan.lpszMenuName     = NULL;  
    classejan.lpszClassName    = szNomeAplic;  
    classejan.hIconSm          = LoadIcon (NULL, IDI_APPLICATION);  
  
    RegisterClassEx (&classejan);  
  
    hjan = CreateWindow (szNomeAplic,      // nome da classe da janela  
                        "Programa Olá",      // titulo da janela  
                        WS_OVERLAPPEDWINDOW, // estilo da janela  
                        CW_USEDEFAULT,       // posicao x inicial  
                        CW_USEDEFAULT,       // posicao y inicial  
                        CW_USEDEFAULT,       // tamanho x inicial  
                        CW_USEDEFAULT,       // tamanho y inicial  
                        NULL,                // handle da janela-mae  
                        NULL,                // handle do menu da janela  
                        hCopia,              // handle da copia do programa  
                        NULL);               // parametros de criacao  
  
    ShowWindow (hjan, iCmdMostrar);  
    UpdateWindow (hjan);
```

Figura 2.1 O programa OLAWIN (*continuação*).

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
HDC hcd;
PAINTSTRUCT ps;
RECT retang;

switch (iMsg)
{
case WM_CREATE:
    PlaySound ("Olawin.wav", NULL, SND_FILENAME | SND_ASYNC);
    // Este arquivo contem a voz do autor, e nao foi traduzido
    return 0;

case WM_PAINT:
    hcd = BeginPaint (hjan, &ps);
    GetClientRect (hjan, &retang);
    DrawText (hcd, "Olá, Windows 95!", -1, &retang,
DT_SINGLELINE | DT_CENTER | DT_VCENTER);
    EndPaint (hjan, &ps);
    return 0;

case WM_DESTROY:
    PostQuitMessage (0);
    return 0;
}

return DefWindowProc (hjan, iMsg, wParam, lParam);
}
```

Figura 2.1 O programa OLAWIN (*continuação*).

No Capítulo 9, você encontrará outro tipo de arquivo comum na programação para o Windows chamado “script de recursos”, com uma extensão .RC. Mas até lá, a maioria dos

programas de exemplo usará simplesmente um arquivo para o Make e um arquivo de código-fonte em C e talvez um arquivo de cabeçalho.

Como mencionei anteriormente, a maior parte de OLAWIN.C é composta por formalidades encontradas em virtualmente todos os programas Windows. Ninguém realmente memoriza toda a sintaxe para escrever essas formalidades; geralmente os programadores iniciam um novo programa copiando um programa existente e fazendo as alterações que forem apropriadas. Sinta-se à vontade para usar dessa maneira os programas do disquete que acompanha o livro.

Se você tem o Windows 95 e o Microsoft Visual C++ 4.0 instalados, e executou o arquivo de lote VCVARS32.BAT incluído no Visual C++ e o arquivo de lote MSC.BAT mostrado no Capítulo 1, será capaz de criar OLAWIN.EXE rodando:

```
NMAKE OLAWIN.MAK
```

na linha de comando do MS-DOS.

Se tudo correr bem, você pode simplesmente rodar o programa a partir da linha de comando do MS-DOS digitando:

```
OLAWIN
```

O programa cria uma janela de aplicativo normal, como mostrado na Figura 2.2. A janela exibe "Olá, Windows 95!" no centro de sua área do cliente. Se você tiver uma placa de som instalada, também ouvirá a mensagem falada (em inglês, com a minha voz!). E se você não tem uma placa de som, o que está esperando para comprar uma?

Pensando nessa janela, você vê que ela tem uma grande quantidade de funções em suas 80 linhas de código. Você pode apanhar a barra de título com o ponteiro do mouse e mover a janela na tela; pode apanhar as molduras e ajustar o tamanho da janela. Alterando o tamanho da janela, o programa reposiciona automaticamente o texto "Olá, Windows 95!" no novo centro da área do cliente. Você pode pressionar o botão de maximização para que a janela ocupe toda a tela; pode pressionar o botão de minimização e reduzir a janela a um ícone. Você pode invocar todas essas opções no menu do sistema. Também pode fechar a janela ao terminar o programa selecionando a opção Fechar no menu do sistema, dando um clique no botão Fechar na extremidade direita da barra de título, ou dando um clique duplo no ícone na extremidade esquerda da barra de título.

Ao mesmo tempo em que você está contente ao ver que o OLAWIN tem a mesma funcionalidade de um programa Windows normal, pode não estar assim tão satisfeito ao ver o tamanho do código-fonte necessário para criá-lo. Tenha, porém, paciência enquanto disseco o programa parte por parte e o analiso até o final.

O Arquivo para o Make

Para facilitar a compilação dos programas Windows, use o utilitário NMAKE, que vem com o Microsoft Visual C++ 4.0. Sempre que você alterar alguma coisa em um dos arquivos-fonte do OLAWIN, tudo o que precisará fazer é executar o NMAKE, como mostrado anteriormente para criar o OLAWIN.EXE executável.

Um arquivo .MAK consiste em uma ou mais seções, cada uma iniciando com uma linha justificada à esquerda que lista o arquivo-alvo, seguida por dois-pontos, e um ou mais arquivos dependentes. Essa linha é seguida por uma ou mais linhas de comando endentadas. Esses comandos criam o arquivo-alvo a partir dos arquivos dependentes. Se alguma data ou hora de modificação de qualquer um dos arquivos dependentes for posterior à data ou hora do arquivo-alvo, então o NMAKE executa as linhas de comando endentadas.

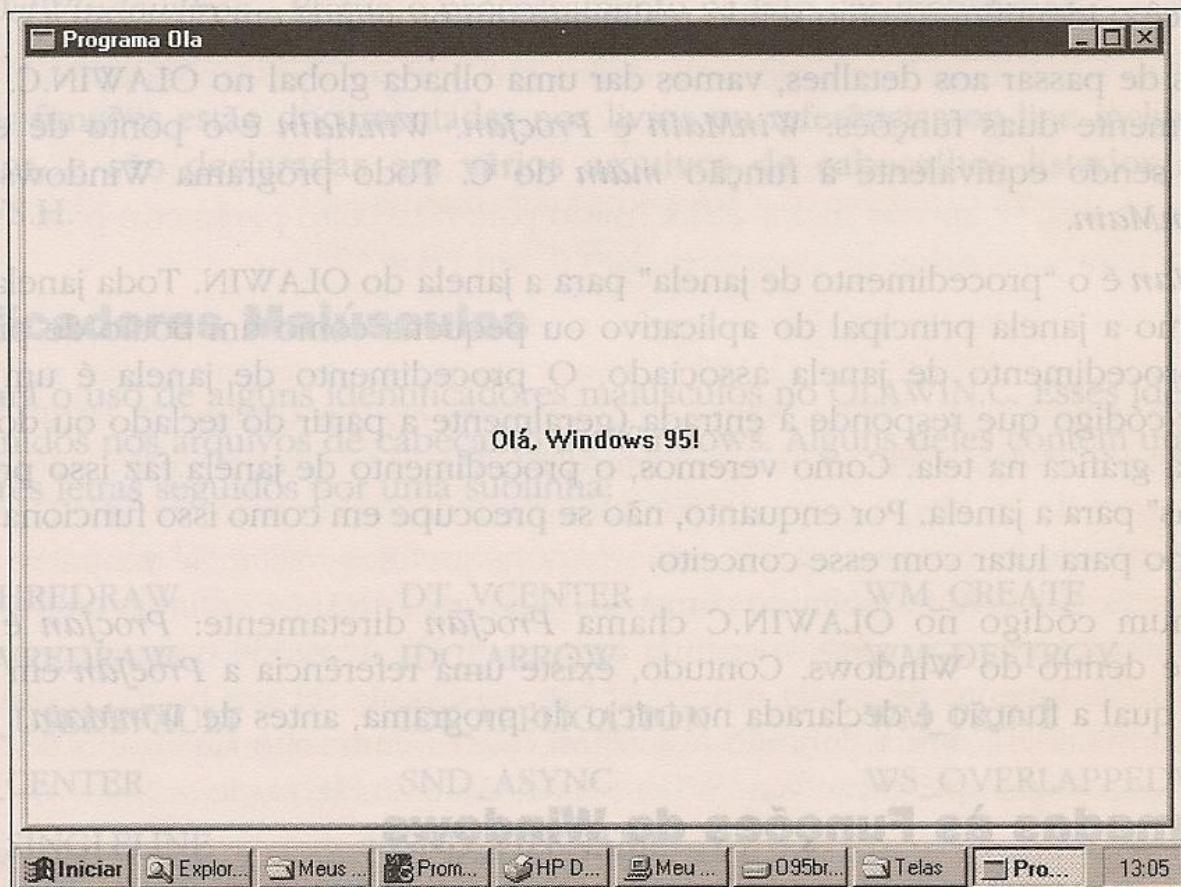


Figura 2.2 O OLAWIN rodando sob o Windows 95.

Normalmente, o NMAKE ou MAKE atualiza somente o arquivo-alvo na primeira seção do arquivo .MAK. Contudo, se um dos arquivos dependentes é ele mesmo um arquivo-alvo em outra seção, então o NMAKE atualiza esse arquivo primeiro.

O arquivo OLAWIN.MAK contém duas seções. A primeira carrega o linker se o OLAWIN.OBJ foi alterado depois do OLAWIN.EXE. A segunda seção roda o compilador C se OLAWIN.C foi alterado depois de OLAWIN.OBJ. Como OLAWIN.OBJ é um arquivo dependente na primeira seção do arquivo .MAK e um alvo na segunda, o utilitário NMAKE verificará se o OLAWIN.OBJ precisa ser atualizado antes de recriar o OLAWIN.EXE. Assim, o arquivo .MAK é na verdade executado do fim para o começo. Rodar o compilador C cria o módulo-objeto OLAWIN.OBJ a partir do arquivo de código-fonte OLAWIN.C. Rodar o linker cria o arquivo executável OLAWIN.EXE a partir de OLAWIN.OBJ.

No Capítulo 1, discuti como os identificadores de macros no arquivo para o MAKE são fornecidos pelas variáveis do ambiente definidas pelos arquivos de lote mostrados naquele

capítulo. Esses identificadores envolvem principalmente a definição de vários sinalizadores do compilador e nomes de bibliotecas do linker, portanto consulte essa seção do Capítulo 1 se estiver interessado em mais detalhes.

O Arquivo do Código-fonte em C

O segundo arquivo na Figura 2.1 é o OLAWIN.C, o código em C. Talvez demore um pouco até que você o identifique como sendo realmente um programa em C!

Antes de passar aos detalhes, vamos dar uma olhada global no OLAWIN.C. O arquivo contém somente duas funções: *WinMain* e *ProcJan*. *WinMain* é o ponto de entrada do programa, sendo equivalente à função *main* do C. Todo programa Windows tem uma função *WinMain*.

ProcJan é o “procedimento de janela” para a janela do OLAWIN. Toda janela – seja ela grande como a janela principal do aplicativo ou pequena como um botão de pressionar – tem um procedimento de janela associado. O procedimento de janela é um modo de encapsular código que responde à entrada (geralmente a partir do teclado ou do mouse) e exibir saída gráfica na tela. Como veremos, o procedimento de janela faz isso processando “mensagens” para a janela. Por enquanto, não se preocupe em como isso funciona. Você terá muito tempo para lutar com esse conceito.

Nenhum código no OLAWIN.C chama *ProcJan* diretamente: *ProcJan* é chamada somente de dentro do Windows. Contudo, existe uma referência a *ProcJan* em *WinMain*, razão pela qual a função é declarada no início do programa, antes de *WinMain*.

As chamadas às Funções do Windows

O programa OLAWIN chama não menos que 17 funções do Windows. Na ordem em que elas aparecem, essas funções (com uma breve descrição) são:

- *LoadIcon* – Carrega um ícone para uso pelo programa.
- *LoadCursor* – Carrega um cursor de mouse para uso pelo programa.
- *GetStockObject* – Obtém um objeto gráfico (neste caso um pincel usado para pintar o fundo da janela).
- *RegisterClassEx* – Registra uma classe de janela para a janela do programa.
- *CreateWindow* – Cria uma janela com base em uma classe de janela.
- *ShowWindow* – Mostra a janela na tela.
- *UpdateWindow* – Faz com que a janela seja redesenhada.
- *GetMessage* – Obtém uma mensagem da fila de mensagem.
- *TranslateMessage* – Traduz algumas mensagens do teclado.
- *DispatchMessage* – Envia uma mensagem a um procedimento de janela.
- *PlaySound* – Reproduz um arquivo de som.

- *BeginPaint* – Inicia a pintura da janela.
- *GetClientRect* – Obtém as dimensões da área do cliente da janela.
- *DrawText* – Mostra uma string de texto.
- *EndPaint* – Finaliza a pintura da janela.
- *PostQuitMessage* – Insere uma mensagem de saída na fila.
- *DefWindowProc* – Efetua o processamento padrão das mensagens.

Essas funções estão documentadas nos livros ou referências on-line incluídos no seu compilador, e são declaradas em vários arquivos de cabeçalhos listados a partir de WINDOWS.H.

Identificadores Maiúsculos

Você notará o uso de alguns identificadores maiúsculos no OLAWIN.C. Esses identificadores estão definidos nos arquivos de cabeçalho do Windows. Alguns deles contêm um prefixo de duas ou três letras seguidos por uma sublinhagem:

CS_HREDRAW	DT_VCENTER	WM_CREATE
CS_VREDRAW	IDC_ARROW	WM_DESTROY
CW_USEDEFAULT	IDI_APPLICATION	WM_PAINT
DT_CENTER	SND_ASYNC	WS_OVERLAPPEDWINDOW
DT_SINGLELINE	SND_FILENAME	

Eles são simplesmente constantes numéricas. O prefixo indica uma categoria geral à qual a constante pertence, como indicado nesta tabela:

Prefixo	Categoria
CS	Opção de classe de estilo
IDI	Número de identificação para um ícone
IDC	Número de identificação para um cursor
WS	Estilo de uma janela
CW	Opção de criar janela
WM	Mensagem de janela
SND	Opção de som
DT	Opção de desenhar texto

Você quase nunca precisará lembrar as constantes numéricas quando estiver programando para o Windows. Quase todas elas têm um identificador definido nos arquivos de cabeçalho.

Novos Tipos de Dados

Alguns outros identificadores usados no OLAWIN.C são tipos de dados novos, também definidos nos arquivos de cabeçalho usando os comandos `typedef` ou `#define`. Isso foi feito originalmente para facilitar a transição dos programas Windows do sistema original de 16 bits para sistemas operacionais futuros que seriam baseados na tecnologia de 32 bits. Isso não funcionou tão bem e de forma tão transparente como todos pensavam na época, mas o conceito era fundamentalmente sólido.

Algumas vezes esses novos tipos de dados são apenas abreviações convenientes. Por exemplo, o tipo de dados `UINT` usado para o segundo parâmetro em `ProcJan` é simplesmente um inteiro não-sinalizado (`unsigned int`, daí a abreviação `UINT`), que no Windows 95 é um valor de 32 bits. O tipo de dados `PSTR` usado para o terceiro parâmetro para `WinMain` é um ponteiro para uma string de caracteres, isto é, um `char *`.

Outros são menos óbvios. Por exemplo, o terceiro e o quarto parâmetro para `ProcJan` estão definidos como `WPARAM` e `LPARAM`, respectivamente. A origem desses nomes requer um pouco de perspectiva histórica. Quando o Windows era um sistema de 16 bits, o terceiro parâmetro para `ProcJan` era definido como `WORD`, um inteiro curto não-sinalizado (`unsigned short`) de 16 bits, e o quarto parâmetro era definido como um `LONG`, um inteiro longo sinalizado de 32 bits, e essa é a razão para os prefixos “W” e “L” na palavra `PARAM`. No Windows 95, `WPARAM` é definido como um `UINT`, e `LPARAM` é definido como um `LONG` (que é simplesmente o tipo `long integer` da linguagem C), de modo que ambos os parâmetros para o procedimento de janela são de 32 bits. Isso pode ser um pouco confuso, pois o tipo de dado `WORD` é ainda definido como um inteiro curto não-sinalizado de 16 bits no Windows 95, de forma que o prefixo “W” em “`PARAM`” é inadequado.

A função `ProcJan` retorna um valor do tipo `LRESULT`. Isso é simplesmente definido como um `LONG`. A função `WinMain` recebe o tipo `WINAPI` (como todas as funções do Windows definidas nos arquivos de cabeçalho), e a função `ProcJan` recebe o tipo `CALLBACK`. Ambos os identificadores são definidos como `_stdcall`, que se refere a uma seqüência de chamada especial para as funções que ocorrem entre o próprio Windows e seu aplicativo.

O OLAWIN também usa quatro estruturas de dados (discutidas posteriormente neste capítulo) definidas nos arquivos de cabeçalho do Windows. São elas:

<i>Estrutura</i>	<i>Significado</i>
<code>MSG</code>	Estrutura da mensagem
<code>WNDCLASSEX</code>	Classe da estrutura da janela
<code>PAINTSTRUCT</code>	Estrutura da pintura
<code>RECT</code>	Estrutura do retângulo

As duas primeiras estruturas de dados são usadas em *WinMain* para definir duas estruturas chamadas *msg* e *classejan*. As duas seguintes são usadas em *ProcJan* para definir duas estruturas chamadas *ps* e *retang*.

Uma Indicação sobre os Handles

Finalmente, existem três identificadores maiúsculos para vários tipos de handles:

Identificador	Significado
HINSTANCE	Handle para uma “ocorrência” – o próprio programa
HWND	Handle para uma janela
HDC	Handle para o contexto de dispositivo

Os handles são usados com bastante freqüência no Windows. Até o fim do capítulo você também encontrará HICON (handle para um ícone), HCURSOR (handle para um cursor do mouse) e HBRUSH (handle para um pincel gráfico).

Um handle é simplesmente um número de 32 bits que referencia um objeto. Os handles no Windows são similares aos handles de arquivo usados na programação convencional em C ou para o MS-DOS. Os programas quase sempre obtêm os handles chamando uma função do Windows. O programa usa o handle ao referenciar o objeto em outras funções do Windows. O valor real de um handle não é importante para o seu programa, mas o módulo do Windows que dá o handle ao seu programa sabe como usá-lo para referenciar o objeto.

Notação Húngara

Você deve ter notado que algumas variáveis no OLAWIN.C têm nomes bem peculiares. Um exemplo é *szLinhaCmd*, passada como parâmetro a *WinMain*.

Muitos programadores Windows usam uma convenção de nomear variáveis conhecida como notação húngara, em honra ao lendário programador Charles Simonyi, da Microsoft. O nome da variável inicia com uma ou mais letras minúsculas, que indicam o tipo de dado da variável. Por exemplo, o prefixo *sz* em *szLinhaCmd* significa “string terminada por zero”. O prefixo *h* em *hCopia* e *hCopiaAnt* significam “handle”; o *i* em *iCmdMostrar* significa “inteiro”. Os dois últimos parâmetros para *ProcJan* também usam a notação húngara, embora, como expliquei anteriormente, *wParam* devesse mais apropriadamente ser chamada de *uiParam* (de *unsigned integer*, inteiro não-sinalizado). Porém, como esses dois parâmetros são definidos usando os tipos de dados WPARAM e LPARAM, preferi conservar seus nomes tradicionais.

Ao nomear variáveis do tipo estrutura, você pode usar o nome da estrutura (ou uma abreviação) em minúsculas como prefixo para o nome da variável ou como todo o nome da variável. Por exemplo, na função *WinMain* em OLAWIN.C, a variável *msg* é uma estrutura do

tipo MSG; *classejan* é uma estrutura do tipo WNDCLASSEX. Na função *ProcJan*, *ps* é uma estrutura PAINTSTRUCT e *retang* é uma estrutura RECT.

A notação húngara ajuda a evitar erros no seu código antes que eles se tornem erros (bugs) no seu programa. Como o nome de uma variável descreve o seu uso e seu tipo de dados, fica mais difícil cometer erros de codificação usando tipos de dados incorretos.

A tabela seguinte mostra os prefixos que usarei neste livro.

<i>Prefixo</i>	<i>Tipo de Dado</i>
<i>c</i>	caractere
<i>by</i>	BYTE (caractere sem sinal)
<i>n</i>	short
<i>i</i>	int
<i>x, y</i>	int (usado como coordenada <i>x</i> ou coordenada <i>y</i>)
<i>cx, cy</i>	int (usado como comprimento <i>x</i> ou <i>y</i> ; o <i>c</i> significa "contador")
<i>b</i> ou <i>f</i>	BOOL (int); <i>f</i> significa flag (sinalizador)
<i>w</i>	WORD (short sem sinal)
<i>l</i>	LONG (tipo long)
<i>dw</i>	DWORD (long sem sinal)
<i>fn</i>	função
<i>s</i>	string
<i>sz</i>	string terminada com o byte 0
<i>h</i>	handle
<i>p</i>	ponteiro

O Ponto de Entrada do Programa

Deixando a visão global para trás, podemos agora iniciar uma dissecação linha a linha do programa. O código inicia com uma declaração `#include` para incluir o arquivo de cabeçalho WINDOWS.H:

```
#include <windows.h>
```

O WINDOWS.H inclui muitos outros arquivos de cabeçalho que contêm as declarações das funções do Windows, as estruturas, os novos tipos de dados e as constantes numéricas.

Em seguida vem uma declaração antecipada da função *ProcJan*:

```
HRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
```

A declaração antecipada é necessária porque *ProcJan* é referenciada dentro da função *WinMain*.

Em um programa C criado para um ambiente convencional, o ponto de entrada é a função *main*. É nessa função que a execução do programa inicia. (Na verdade, a função *main* é o ponto de entrada para a parte do programa desenvolvida pelo programador.

Geralmente o compilador C insere um código de inicialização no arquivo executável e esse código de inicialização chama a função *main*.) O ponto de entrada de um programa Windows é uma função chamada *WinMain*. Ela é sempre definida da seguinte maneira:

```
int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt,
                     PSTR szLinhaCmd, int iCmdMostrar)
```

Essa função usa uma seqüência de chamada WINAPI e retorna um inteiro ao Windows quando termina. A função precisa ser chamada *WinMain*. Ela tem quatro parâmetros.

O parâmetro *hCopia* é o “handle da cópia”, ou seja, o número que identifica o programa quando ele está em execução sob o Windows. Talvez o usuário esteja executando múltiplas cópias (ou ocorrências) do mesmo programa no Windows. Cada cópia tem um valor *hCopia* diferente. O handle da cópia pode ser comparado a um “identificador da tarefa” ou “identificador do processo” comum em sistemas operacionais de multitarefa.

O parâmetro *hCopiaAnt* (“ocorrência anterior”) agora é obsoleto. Nas versões anteriores do Windows, ele referenciava o handle da ocorrência mais recente do programa que ainda estava ativa. Se não existissem outras cópias carregadas, o valor de *hCopiaAnt* seria 0, ou NULL. Sob o Windows 95, esse parâmetro é sempre NULL.

O parâmetro *szLinhaCmd* é um ponteiro para uma string terminada por 0, que contém os parâmetros da linha de comando passados ao programa. É possível executar um programa Windows com um parâmetro na linha de comando digitando o nome do programa e o parâmetro na linha de prompt do MS-DOS, ou digitando o nome do programa e o parâmetro na caixa de diálogo Executar, invocada a partir do menu Iniciar.

O parâmetro *iCmdMostrar* é um número que indica como a janela deverá ser inicialmente mostrada no Windows. Geralmente, os programas não precisam examinar esse número, mas podem fazê-lo, se quiserem. Na maioria dos casos, o número é 1 ou 7. Todavia, é melhor não pensar nele como sendo 1 ou 7. Pense no valor como SW_SHOWNORMAL (definido em WINDOWS.H como 1) ou SW_SHOWMINNOACTIVE (definido como 7). O prefixo SW nesses identificadores é uma sigla para “mostrar janela” (show window). O valor indica se o usuário carregou o programa para ser mostrado como uma janela normal ou para estar inicialmente minimizado.

Registrando a Classe da Janela

Uma janela é sempre criada com base em uma classe. A classe da janela identifica o procedimento de janela que processa as mensagens. Como isso é importante, vou repetir: uma janela é sempre criada com base em uma classe. A classe da janela identifica o procedimento de janela que processa as mensagens.

Mais de uma janela pode ser criada com base em uma única classe. Por exemplo, todos os botões de janela no Windows são criados com base na mesma classe. A classe da janela define o procedimento de janela e algumas outras características das janelas que são criadas com base nesta classe. Ao criar uma janela, você define características adicionais que são específicas àquela janela.

Antes de criar uma janela para seu programa, você precisa registrar uma classe de janela chamando a função *RegisterClassEx*. Essa é uma versão estendida (daí o sufixo “Ex”) da função *RegisterClass* encontrada nas versões anteriores do Windows. Porém, *RegisterClass* ainda funciona sob o Windows 95.

A função *RegisterClassEx* requer um único parâmetro: um ponteiro para uma estrutura do tipo **WNDCLASSEX**, que está definida nos arquivos de cabeçalho do Windows da seguinte maneira:

```
typedef struct tagWNDCLASSEX
{
    UINT         cbSize;
    UINT         style;
    WNDPROC     lpfnWndProc;
    int          cbClsExtra;
    int          cbWndExtra;
    HINSTANCE   hInstance;
    HICON        hIcon;
    HCURSOR     hCursor;
    HBRUSH      hbrBackground;
    LPCSTR      lpszMenuName;
    LPCSTR      lpszClassName;
    HICON        hIconSm;
}
WNDCLASSEX;
```

Algumas observações sobre os tipos de dados e a notação húngara aqui: Os prefixos *LP* e *lp* significam “ponteiro long”, que são remanescentes do Windows de 16 bits, onde os programadores precisavam distinguir entre ponteiros short (ou near) de 16 bits e long (ou far). No Windows 95, todos os ponteiros têm 32 bits de tamanho. Procurei remover todos os prefixos *l* nos tipos de ponteiro nos programas de exemplo deste livro, mas você certamente os verá em outros locais.

Observe também outros usos da notação húngara: O prefixo *lpfn* significa “ponteiro long a uma função”. O prefixo *cb* significa “contagem de bytes”. O prefixo *hbr* significa “handle para um pincel”.

Em *WinMain*, você precisa definir uma estrutura do tipo **WNDCLASSEX**, geralmente assim:

```
WNDCLASSEX classejan;
```

Em seguida, defina os 12 campos da estrutura e chame *RegisterClassEx*:

```
RegisterClassEx (&classejan);
```

Os dois campos mais importantes são o penúltimo e o terceiro. O penúltimo é o nome da classe de janela (que nos programas que criam apenas uma janela geralmente é o próprio nome do programa). O terceiro campo (*lpfnWndProc*) é o endereço do procedimento de janela usado para todas as janelas criadas com base nessa classe (que é a função *ProcJan* no

OLAWIN.C). Os demais campos descrevem características de todas as janelas com base nessa classe.

O campo *sbSize* é o tamanho da estrutura. A declaração:

```
classejan.style = CS_HREDRAW | CS_VREDRAW;
```

combina dois identificadores de “estilo de classe” com o operador OU bit a bit da linguagem C. Nos arquivos de cabeçalho do WINDOWS, os diversos identificadores que iniciam com o prefixo CS são definidos como constantes de 32 bits com um bit ligado. Por exemplo, CS_VREDRAW é definido como 0x0001, e CS_HREDRAW como 0x0002. Os identificadores definidos dessa forma são às vezes chamados “sinalizadores de bit”. Para combinar os identificadores de um bit, usa-se o operador OU bit a bit da linguagem C.

Esses dois identificadores do estilo da classe indicam que todas as janelas criadas com base nessa classe deverão ser completamente repintadas sempre que o tamanho horizontal da janela (CS_HREDRAW) ou o tamanho vertical (CS_VREDRAW) for alterado. Se você alterar o tamanho da janela do OLAWIN, verá que o texto será repositionado para ficar no novo centro da janela. Esses dois identificadores garantem que isso aconteça. Você verá em breve como o procedimento de janela é notificado dessa alteração no tamanho da janela.

O terceiro campo da estrutura WNDCLASSEX é inicializado pela seguinte declaração:

```
classejan.lpfnWndProc = ProcJan;
```

Isso define o procedimento de janela para essa janela como *ProcJan*, a segunda função em OLAWIN.C. Esse procedimento de janela processará todas as mensagens para todas as janelas criadas com base nessa classe de janela. Como mencionei, o prefixo *lpfn* no nome do campo é a notação húngara para “ponteiro long a uma função”.

As duas declarações seguintes:

```
classejan.cbClsExtra = 0;
```

```
classejan.cbWndExtra = 0;
```

reservam um espaço extra na estrutura da classe e na estrutura da janela que o Windows mantém internamente. Um programa pode usar esse espaço extra para um propósito próprio. O OLAWIN não utiliza esse recurso, e portanto é especificado zero. Caso contrário, como indica a notação húngara, o campo seria definido com uma contagem ou um número de bytes.

O próximo campo é simplesmente o handle da cópia do programa (que é um dos parâmetros para *WinMain*):

```
classejan.hInstance = hCopia;
```

A declaração:

```
classejan.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

e a declaração:

```
classejan.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
```

definem um ícone para todas as janelas criadas com base nessa classe de janela. O ícone é uma pequena figura em mapa de bits que aparece na barra de tarefas do Windows e na extremidade esquerda da barra de título. Posteriormente neste livro, você aprenderá a criar ícones personalizados para os seus programas Windows. Por enquanto, vamos utilizar um ícone predefinido.

Para obter um handle a um ícone predefinido, chame a função *LoadIcon* com o primeiro parâmetro deixado como NULL. (Quando você carregar seu próprio ícone personalizado, esse parâmetro será igual ao handle da cópia do programa.) O segundo parâmetro é um identificador que inicia com IDI (identificação de um ícone) definido nos arquivos de cabeçalho do WINDOWS. O ícone IDI_APPLICATION é simplesmente uma pequena imagem de uma janela. A função *LoadIcon* retorna um handle a esse ícone. Na verdade não precisamos preocupar-nos com o valor desse handle. Ele é simplesmente usado para definir o valor dos campos *hIcon* e *hIconSm*. O campo *hIcon* está definido na estrutura WNDCLASSEX como sendo do tipo HICON, o que significa “handle de um ícone”.

A declaração:

```
classejan.hCursor = LoadCursor (NULL, IDC_ARROW);
```

é muito similar às anteriores. A função *LoadCursor* carrega um cursor predefinido do mouse conhecido como IDC_ARROW e retorna um handle para o cursor. Esse handle é atribuído ao campo *hCursor* da estrutura WNDCLASSEX. Quando o cursor do mouse aparece na área do cliente de uma janela criada com base nessa classe, o cursor transforma-se em uma pequena seta.

O próximo campo especifica a cor de fundo da área do cliente das janelas criadas com base nessa classe. O prefixo *hbr* no nome de campo *hbrBackground* significa “handle de um pincel”. Pincel é um termo gráfico que referencia um padrão colorido de pontos usados para preencher uma área. O Windows tem diversos pincéis padrão. A chamada à função *GetStockObject*, mostrada aqui, retorna um handle para um pincel branco:

```
classejan.hbrBackground = GetStockObject (WHITE_BRUSH);
```

Isso significa que o fundo da área do cliente da janela será branco, o que é uma escolha comum.

O próximo campo especifica o menu da classe de janela. O OLAWIN não tem um menu e, por isso, o campo recebe o valor NULL:

```
classejan.lpszMenuName = NULL;
```

Finalmente, a classe precisa receber um nome. Para um programa pequeno, pode ser simplesmente o nome do programa, que é a string “OLAWIN”, armazenada na variável *szNomeAplic*:

```
classejan.lpszClassName = szNomeAplic;
```

Após a inicialização dos 12 campos da estrutura, o OLAWIN registra a classe da janela chamando a função *RegisterClassEx*. O único parâmetro para a função é um ponteiro à estrutura WNDCLASS:

```
RegisterClassEx (&classejan);
```

Criando a Janela

A classe da janela define as características gerais de uma janela, permitindo assim que a mesma classe seja usada para criar muitas janelas diferentes. Quando você cria uma janela chamando *CreateWindow*, especifica informações mais detalhadas sobre ela.

Os programadores iniciantes no Windows algumas vezes ficam confusos com a distinção entre classe da janela e janela, e por que todas as características de uma janela não podem ser especificadas de uma só vez. Na verdade, dividir as informações dessa maneira é muito conveniente. Por exemplo, todas as janelas de botão de apertar são criadas com base na mesma classe de janela. O procedimento de janela associado a essa classe de janela está localizado dentro do próprio Windows. A classe de janela é responsável pelo processamento da entrada feita pelo teclado e pelo mouse ao botão de apertar e por definir a aparência visual do botão na tela. Todos os botões de pressionar funcionam da mesma forma nesse ponto. Porém, nem todos os botões de apertar são iguais. Eles podem ter tamanhos diferentes, localizações diferentes na tela, e diferentes strings de texto. Essas últimas características são parte da definição da janela.

Em vez de usar a estrutura de dados como faz *RegisterClassEx*, a função *CreateWindow* requer que todas as informações sejam passadas como parâmetros à função. Aqui está a chamada à *CreateWindow* no OLAWIN.C:

```
hjan = CreateWindow (szNomeAplic, // nome da classe da janela
                     "Programa Olá", // titulo da janela
                     WS_OVERLAPPEDWINDOW, // estilos da janela
                     CW_USEDEFAULT, // posição x inicial
                     CW_USEDEFAULT, // posição y inicial
                     CW_USEDEFAULT, // tamanho x inicial
                     CW_USEDEFAULT, // tamanho y inicial
                     NULL, // handle da janela-mae
                     NULL, // handle do menu da janela
                     hCopia, // handle da cópia do programa
                     NULL); // parametros de criação
```

Para sua conveniência, utilizei o símbolo // para os comentários de uma única linha para descrever os parâmetros para a função *CreateWindow*.

O parâmetro chamado “nome da classe da janela” é *szNomeAplic*, que contém a string “Olá, Windows” – o nome da classe de janela que acabamos de registrar. É assim que a janela é associada à classe.

A janela criada por esse programa é uma janela normal sobreposta, com uma barra de título, uma caixa do menu de sistema à esquerda da barra de título, botões de maximização,

de minimização e de fechamento à direita da barra de título, e uma moldura para a alteração do tamanho. Esse é um estilo normal de janela e tem o nome WS_OVERLAPPEDWINDOW, que aparece como o parâmetro “estilo da janela”. O “título da janela” é o texto que aparecerá na barra de título.

Os parâmetros identificados como “posicao x inicial” e “posicao y inicial” especificam a posição inicial do canto superior esquerdo da janela relativa ao canto superior esquerdo da tela. Usando o identificador CS_USEDEFAULT para esses parâmetros, estamos indicando que queremos que o Windows use a posição padrão para uma janela sobreposta. (CW_USEDEFAULT está definido com 0x80000000.) Por padrão, o Windows posiciona sucessivas janelas sobrepostas ligeiramente deslocadas horizontal e verticalmente a partir do canto superior esquerdo da tela. Similarmente, os parâmetros “tamanho x inicial” e “tamanho y inicial” especificam a largura e a altura da janela. O identificador CS_USEDEFAULT novamente indica que queremos que o Windows use um tamanho padrão para a janela.

O parâmetro identificado como “handle da janela-mae” está com NULL porque essa janela não tem uma mãe. (Quando existe o relacionamento mãe-filha entre duas janelas, a filha sempre aparece ocupando uma área dentro da mãe.) O “handle do menu da janela” está em NULL porque a janela não tem um menu. O “handle da cópia do programa” está associado ao handle da cópia passado ao programa como um parâmetro de *WinMain*. Finalmente, um ponteiro “parametros de criacao” está definido como NULL. Você poderia usá-lo para acessar dados que mais tarde quisesse referenciar no programa.

A chamada a *CreateWindow* retorna um handle para a janela criada. Esse handle é colocado na variável *hjan*, que está definida como do tipo HWND (handle de uma janela). Toda janela no Windows tem um handle. O seu programa usa o handle para referenciar a janela. Muitas funções no Windows requerem *hjan* como parâmetro, para que o Windows saiba em que janela a função opera. Quando um programa cria muitas janelas, cada uma tem um handle diferente. O handle de uma janela é um dos handles mais importantes com que um programa Windows trata.

Exibindo a Janela

Após o retorno da função *CreateWindow*, a janela foi criada internamente no Windows. Entretanto, ela ainda não aparece na tela. Duas outras chamadas são necessárias. A primeira é:

```
ShowWindow (hjan, iCmdMostrar);
```

O primeiro parâmetro é o handle à janela que acabou de ser criada por *CreateWindow*. O segundo é o valor *iCmdMostrar* passado como parâmetro à *WinMain*. Isso determina como a janela será inicialmente mostrada na tela. Se esse valor for SW_SHOWNORMAL (igual a 1), a janela será mostrada normalmente; se for SW_SHOWMINNOACTIVE (igual a 7), então a janela não será exibida, mas seu nome e seu ícone aparecerão na barra de tarefas.

A função *ShowWindow* coloca a janela na tela. Se o segundo parâmetro para *ShowWindow* for SW_SHOWNORMAL, a área do cliente da janela será apagada com o pincel do fundo especificado na classe de janela. A chamada à função:

```
UpdateWindow (hjan);
```

faz com que a área do cliente seja pintada. Ela efetua isso enviando ao procedimento de janela (a função *ProcJan* no OLAWIN.C) uma mensagem WM_PAINT. Logo examinaremos como *ProcJan* trata dessa mensagem.

O Laço de Mensagem

Após a chamada a *UpdateWindow*, a janela fica totalmente visível na tela. O programa precisa agora preparar-se para ler as entradas do usuário pelo teclado e pelo mouse. O Windows mantém uma “fila de mensagens” para cada programa Windows em execução. Quando ocorre o evento de uma entrada, o Windows traduz o evento para uma “mensagem” que coloca na fila de mensagens do programa.

O programa retira essas mensagens da fila executando um bloco de código chamado de “laço de mensagem”:

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg.wParam;
```

A variável *msg* é uma estrutura do tipo MSG, que está definida nos arquivos de cabeçalho do WINDOWS como segue:

```
typedef struct tagMSG
{
    HWND      hwnd;
    UINT      message;
    WPARAM    wParam;
    LPARAM    lParam;
    DWORD     time;
    POINT    pt;
}
MSG;
```

O tipo de dado POINT é, porém, outra estrutura, definida assim:

```
typedef struct tagPOINT
{
    LONG x;
    LONG y;
}
POINT;
```

A chamada a *GetMessage* que inicia o laço retira uma mensagem da fila:

```
GetMessage (&msg, NULL, 0, 0);
```

Essa chamada passa ao Windows um ponteiro à estrutura MSG chamada *msg*. O segundo, o terceiro e o quarto parâmetros estão como NULL ou 0 para indicar que o programa quer todas as mensagens para todas as janelas criadas pelo programa. O Windows preenche os campos da estrutura da mensagem com a próxima mensagem da fila. Os campos nessa estrutura são:

- *bund* – O handle para a janela à qual a mensagem está direcionada. No programa OLAWIN, isso é o mesmo que o valor *bjan* retornado por *CreateWindow*, pois essa é a única janela que o programa tem.
- *message* – O identificador da mensagem. É um número que identifica a mensagem. Para cada mensagem, existe um identificador correspondente definido nos arquivos de cabeçalho do WINDOWS que inicia com o prefixo WM (“mensagem da janela”, ou window message). Por exemplo, se você posicionar o ponteiro do mouse na área do cliente do OLAWIN e pressionar o botão esquerdo, o Windows colocará uma mensagem na fila com um campo *message* igual a WM_LBUTTONDOWN, que tem o valor 0x0201.
- *wParam* – Um “parâmetro da mensagem” de 32 bits, cujo significado e valor dependem da mensagem particular.
- *lParam* – Outro parâmetro de 32 bits que depende da mensagem.
- *time* – A hora em que a mensagem foi colocada na fila.
- *pt* – As coordenadas do mouse na hora em que a mensagem foi colocada na fila.

Se o campo *message* da mensagem retirada da fila é qualquer coisa exceto WM_QUIT (que é igual a 0x0012), então *GetMessage* retorna um valor diferente de zero. Uma mensagem WM_QUIT faz com que o programa saia do laço de mensagem. O programa então termina, retornando o membro *wParam* da estrutura *msg*.

A declaração:

```
TranslateMessage (&msg);
```

passa a estrutura *msg* de volta ao Windows para alguma tradução do teclado. (Isso será mais discutido no Capítulo 5.) A declaração:

```
DispatchMessage (&msg);
```

novamente passa a estrutura *msg* de volta ao Windows, que então envia a mensagem ao procedimento de janela apropriado para processamento – isto é, o Windows chama o procedimento de janela. Esse procedimento no programa OLAWIN.C é a função *ProcJan*. Após o processamento da mensagem, ela retorna ao Windows, que ainda está no serviço da chamada à função *DispatchMessage*. Quando o Windows retorna ao OLAWIN seguindo a chamada a *DispatchMessage*, o laço da mensagem continua com a próxima chamada à *GetMessage*.

O Procedimento de Janela

Tudo o que descrevemos até aqui foram, na verdade, apenas formalidades e preparativos. A classe foi registrada, a janela foi criada e apresentada na tela, e o programa entrou no laço de mensagens para retirar as mensagens da fila.

A ação real ocorre no procedimento de janela, que determina o que a janela apresenta em sua área do cliente e como ela responde às entradas do usuário.

No OLAWIN, o procedimento de janela é a função chamada *ProcJan*. O procedimento de janela pode ter qualquer nome (logicamente, desde que não entre em conflito com outro nome). Um programa Windows pode conter mais de um procedimento de janela. Um procedimento de janela sempre está associado a determinada classe de janela que você registra chamando *RegisterClassEx*. A função *CreateWindow* cria uma janela com base em determinada classe de janela. Mais de uma janela pode ser criada com base na mesma classe de janela.

Um procedimento de janela sempre é definido assim:

```
LRESULT CALLBACK ProcJan ( HWND hjan, UINT iMsg,
                           WPARAM wParam, LPARAM lParam)
```

Note que os quatro parâmetros para o procedimento de janela são idênticos aos quatro primeiros campos da estrutura MSG.

O primeiro parâmetro é *hjan*, o handle para a janela que está recebendo a mensagem. Esse é o mesmo handle retornado da função *CreateWindow*. Para um programa como o OLAWIN, que cria somente uma janela, esse é o único handle de janela que o programa conhece. Quando um programa cria múltiplas janelas com base na mesma classe (e portanto o mesmo procedimento de janela), então *hjan* identifica a janela particular que está recebendo a mensagem.

O segundo parâmetro é um número (especificamente, um número inteiro não-sinalizado de 32 bits, um *UINT*) que identifica a mensagem. Os dois últimos parâmetros (um *WPARAM* chamado *wParam* e um *LPARAM* chamado *lParam*) fornecem mais informações sobre a mensagem. Eles são conhecidos como “parâmetros da mensagem”. O que esses parâmetros contêm é específico para cada tipo de mensagem.

Processando as Mensagens

Cada mensagem que um procedimento de janela recebe é identificado por um número, que é o parâmetro *iMsg* para o procedimento. Os arquivos de cabeçalho do WINDOWS definem identificadores iniciando com WM (window message, ou “mensagem da janela”) para cada parâmetro da mensagem.

Geralmente, os programadores usam uma construção *switch* e *case* para determinar qual mensagem o procedimento de janela está recebendo e como processá-la corretamente. Todas as mensagens que um procedimento de janela escolher não processar precisam ser passadas à função *DefWindowProc*. O valor devolvido por *DefWindowProc* precisa ser retornado dentro do procedimento de janela.

No OLAWIN, *ProcJan* escolhe processar somente três mensagens: WM_CREATE, WM_PAINT e WM_DESTROY. O procedimento de janela está estruturado assim:

```

switch (mensagem)
{
    case WM_CREATE:
        [processa a mensagem WM_CREATE]
        return 0;

    case WM_PAINT:
        [processa a mensagem WM_PAINT]
        return 0;

    case WM_DESTROY:
        [processa a mensagem WM_DESTROY]
        return 0;
}

return DefWindowProc (hjan, mensagem, wParam, lParam);

```

É essencial chamar *DefWindowProc* para efetuar o processamento padrão de todas as mensagens que o seu procedimento de janela não processar.

Reproduzindo um Arquivo de Som

A primeira mensagem que um procedimento de janela recebe – e a primeira que *ProcJan* escolhe processar – é WM_CREATE. *ProcJan* recebe essa mensagem enquanto o Windows está processando a função *CreateWindow* em *WinMain*. Isto é, quando o OLAWIN chama *CreateWindow*, o Windows faz o que tem de fazer e, no processo, chama *ProcJan* com o primeiro parâmetro definido com o handle da janela e o segundo definido com WM_CREATE. *ProcJan* processa a mensagem WM_CREATE e retorna o controle de volta para o Windows. O Windows então pode retornar de *CreateWindow* para o OLAWIN a fim de continuar o processamento em *WinMain*.

Geralmente um procedimento de janela efetua a inicialização da janela uma vez durante o processamento de WM_CREATE. OLAWIN escolhe processar essa mensagem reproduzindo um arquivo de som de forma de onda chamado OLAWIN.WAV. Ele faz isso usando a função *PlaySound*. O primeiro parâmetro é o nome do arquivo. Ele também poderia ser um alias do som definido na seção Sons do Painel de Controle, ou um recurso do programa. O segundo parâmetro é usado somente se o arquivo de som for um recurso. O terceiro parâmetro especifica algumas opções. Neste caso, indicamos que o primeiro parâmetro é um nome de arquivo, e o som deve ser reproduzido de modo assíncrono, isto é, a função *PlaySound* deve retornar assim que o arquivo de som começar a ser reproduzido, sem esperar que ele termine.

ProcJan conclui o processamento de WM_CREATE, retornando 0 do procedimento de janela.

A Mensagem WM_PAINT

A segunda mensagem que *ProcJan* processa é WM_PAINT. Ela é extremamente importante na programação para o Windows, pois informa ao programa quando parte ou toda a área do cliente da janela está “inválida” e precisa ser repintada.

Como a área do cliente torna-se inválida? Assim que a janela é criada, toda a área do cliente é inválida, pois o programa ainda não desenhou nada na janela. A primeira mensagem WM_PAINT (que normalmente ocorre quando o programa chama *UpdateWindow* em *WinMain*) direciona o procedimento de janela a desenhar algo na área do cliente.

Quando você altera o tamanho da janela do OLAWIN, a área do cliente também se torna inválida. Lembre-se de que o parâmetro *style* na estrutura *classejan* do nosso programa foi definido com os valores CS_HREDRAW e CS_VREDRAW. Isso direciona o Windows a invalidar toda a janela quando o tamanho for alterado. O procedimento de janela recebe uma mensagem WM_PAINT.

Quando você minimiza o OLAWIN e depois restaura a janela novamente ao seu tamanho anterior, o Windows não salva o conteúdo da área do cliente. Em um ambiente gráfico, isso representaria um volume de dados muito grande. Em vez disso, o Windows invalida a janela. O procedimento de janela recebe uma mensagem WM_PAINT e ele mesmo restaura o conteúdo da sua janela.

Quando você move as janelas e elas se sobreponem, o Windows não salva a área de uma janela que foi coberta por outra. Quando essa área da janela mais tarde é descoberta, a janela é sinalizada como inválida. O procedimento de janela recebe uma mensagem WM_PAINT para repintar o conteúdo da janela.

O processamento de WM_PAINT quase sempre inicia com uma chamada à função *BeginPaint*:

```
hcd = BeginPaint (hjan, &ps);
e termina com uma chamada a EndPaint:
EndPaint (hjan, &ps);
```

Em ambos os casos, o primeiro parâmetro é um handle à janela do programa e o segundo é um ponteiro para uma estrutura do tipo PAINTSTRUC. PAINTSTRUC contém algumas informações que um procedimento de janela pode usar para pintar a área do cliente. (Os campos dessa estrutura serão discutidos no próximo capítulo.)

Durante a chamada a *BeginPaint*, o Windows apaga o fundo da área do cliente caso ele ainda não tenha sido apagado. Ele apaga o fundo usando o pincel especificado no campo *hbrBackground* da estrutura WNDCLASSEX usada para registrar a classe da janela. No caso do OLAWIN, esse é um pincel branco, o que significa que o Windows apaga o fundo pintando-o de branco. A função *BeginPaint* valida toda a área do cliente e retorna o “handle a um contexto do dispositivo”. O contexto do dispositivo refere-se a um dispositivo de saída (como a tela do monitor) e ao seu controlador de dispositivo. Você precisa do handle do contexto do dispositivo para exibir texto e gráficos na área do cliente de uma janela. Usando

o handle do contexto do dispositivo retornado por *BeginPaint*, você não pode desenhar fora da área do cliente, mesmo que tente. *EndPaint* libera o handle do contexto do dispositivo, de modo que ele deixa de ser válido.

Se um procedimento de janela não processar as mensagens WM_PAINT (o que é muito raro), elas precisam ser passadas para *DefWindowProc*, que simplesmente chama *BeginPaint* e *EndPaint*, uma após a outra, para que a área do cliente seja validada.

Depois que a função *ProcJan* chamar *BeginPaint*, ela chama *GetClientRect*:

```
GetClientRect (hjan, &retang);
```

O primeiro parâmetro é o handle da janela do programa. O segundo é um ponteiro para uma variável chamada *retang*, definida como do tipo RECT em *ProcJan*.

RECT é uma estrutura “retângulo” definida nos arquivos de cabeçalho do WINDOWS. Ela tem quatro campos LONG chamados *left*, *top*, *right* e *bottom*. *GetClientRect* define esses quatro campos com as dimensões da área do cliente da janela. Os campos *left* e *top* são sempre deixados em 0. Os campos *right* e *bottom* são definidos com a largura e a altura em pixels da área do cliente.

ProcJan não faz nada com essa estrutura RECT, exceto passar um ponteiro para ela como o quarto parâmetro de *DrawText*:

```
DrawText (hcd, "Ola, Windows 95!", -1, &retang,  
          DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

DrawText (como o nome implica) desenha texto. Como essa função desenha algo, o primeiro parâmetro é o handle do contexto do dispositivo retornado por *BeginPaint*. O segundo é o texto a desenhar, e o terceiro é o valor -1, para indicar que a string de texto termina com um byte 0.

O último parâmetro é uma série de bits sinalizadores definidos nos arquivos de cabeçalho do WINDOWS. Os sinalizadores indicam que o texto deve ser mostrado como uma única linha centralizada horizontal e verticalmente dentro do retângulo especificado pelo quarto parâmetro. A chamada a essa função faz assim com que a string “Ola, Windows 95!” apareça no centro da área do cliente.

Sempre que a área do cliente se torna inválida (como acontece quando você altera o tamanho da janela), *ProcJan* recebe uma nova mensagem WM_PAINT. *ProcJan* obtém o tamanho da janela atualizada chamando *GetClientRect* e novamente exibe o texto no novo centro da janela.

A Mensagem WM_DESTROY

WM_DESTROY é outra mensagem importante. Ela indica que o Windows está no processo de destruir uma janela devido a um comando do usuário. A mensagem resulta da escolha pelo usuário do botão Fechar, da opção Fechar no menu de sistema do programa, ou por meio do comando de teclado Alt-F4.

O OLAWIN responde a essa mensagem na forma normal, chamando:

```
PostQuitMessage (0);
```

Essa função insere uma mensagem WM_QUIT na fila do programa. Mencionei anteriormente que *GetMessage* retorna 0 para qualquer mensagem diferente de WM_QUIT que retira da fila. Quando a função *GetMessage* retira uma mensagem WM_QUIT, ela retorna 0. Isso faz com que *WinMain* abandone o laço de mensagem e saia, encerrando o programa.

DIFÍCULDADES NA PROGRAMAÇÃO PARA O WINDOWS

Mesmo com as minhas explicações sobre o OLAWIN, a estrutura e o funcionamento do programa provavelmente continuam um pouco obscuros. Em um pequeno programa em C para um ambiente convencional, todo o programa precisa estar dentro da função *main*. No OLAWIN, *WinMain* contém somente as formalidades necessárias para registrar a classe da janela, criar a janela, e retirar e despachar as mensagens da fila.

Toda a ação real do programa ocorre no procedimento de janela. No OLAWIN, essa ação é bem simples – colocar uma string de texto em uma janela. Em outros capítulos você verá que quase tudo o que um programa faz é em resposta a uma mensagem do procedimento de janela. Esse é um dos principais obstáculos conceituais que você precisa transpor, antes de começar a desenvolver programas para o Windows.

Não Me Chame, Eu Chamarei Você

Como mencionei anteriormente, os programadores estão habituados à idéia de chamar o sistema operacional para executar determinadas tarefas. Por exemplo, quem programa em C usa a função *open* ou *fopen* para abrir um arquivo. As funções de biblioteca, por sua vez, chamam o sistema operacional para abrir os arquivos.

Entretanto, o Windows é diferente. Embora ele tenha mais de 1.000 funções que o seu programa pode chamar, o Windows também chama o seu programa, especificamente através do procedimento de janela que denominamos *ProcJan*. O procedimento de janela está associado a uma classe que o programa registra chamando *RegisterClassEx*. Uma janela criada com base nessa classe usa esse procedimento para processar todas as mensagens para ela. O Windows envia uma mensagem para a janela chamando o procedimento de janela.

O Windows chama *ProcJan* assim que uma janela for criada, e também quando ela for destruída mais tarde; ele chama *ProcJan* quando o tamanho da janela for alterado; quando ela for movida ou reduzida a um ícone; quando um item for selecionado no menu; quando a barra de rolagem for movida ou selecionada com o mouse; quando precisar avisá-la de que é necessário repintar a área do cliente.

Todas essas chamadas são na forma de mensagens. Na maioria dos programas Windows, a principal e maior seção do programa é dedicada ao tratamento dessas mensagens. As aproximadamente 200 mensagens diferentes que o Windows pode enviar ao procedimento de janela são todas identificadas com nomes que iniciam com as letras WM e definidas nos arquivos de cabeçalho do Windows.