

# Apostila

## Programação Aplicada em C para Windows

**Eduard Montgomery Meira Costa, DSc**

2002

## 1. INTRODUÇÃO

A necessidade do conhecimento em programação em C para Windows tem se tornado crescente à medida em que a tecnologia evolui. Em todas as áreas do conhecimento, programas são criados para realizar as mais variadas formas de análise, como estatística, comércio, simulação, imagem, internet, etc.

Especificamente na área de engenharia, os computadores têm se tornado indispensáveis. Tanto para cálculos, como para simulação ou desenvolvimento de projetos, o computador tem que estar presente. Programas computacionais são necessários, especialmente para simulação de processos, os quais eliminam problemas de custos e inconveniência da manipulação real, através de modelos.

A plataforma Windows é de grande importância no mundo atual, desde que é um dos sistemas operacionais mais utilizados. Sua interface é de ótima aparência. Vários recursos se apresentam para facilitar a operação de um programa por parte do usuário, bem como para o programador: botões, mouse, gráficos, texto, menu, etc.

Naturalmente, a programação para Windows exige dedicação e paciência. São mais de 1000 comandos específicos para trabalhar com todos os recursos do Windows. Entretanto, o Windows apresenta funções especiais para o acesso direto à periféricos, como monitor de vídeo, impressoras, disco rígido e flexível, e o que se torna necessário é o aprendizado das funções básicas para se compreender como o Windows gerencia um programa, como são processadas suas mensagens, seus procedimentos de janela, entre outros, e saber criar uma janela e desenvolver o que se deseja com os recursos necessários. Ademais, livros específicos sobre C para Windows devem ser consultados para aumentar o conhecimento de outras funções. Para isto, o programador deve ter conhecimento da linguagem C. Várias funções e formatos de procedimentos devem ser gerados em uma seqüência, tal que sejam compreendidos pelo computador, de forma a abrir uma janela e realizar as funções desejadas.

A metodologia utilizada aqui é fundamentada sobre a construção de um programa para Windows na linguagem C, onde vários recursos são utilizados: ícone, mouse, menu, teclado, teclas aceleradoras, cores, gráficos, movimentos, gravação e abertura de arquivos, impressão, caixas de mensagens, caixas de diálogo, temporizadores, entre outros, onde cada função e cada mensagem WINDOWS, cada estrutura e cada termo utilizado é explicado para assegurar que o leitor não se perca no desenvolvimento e execução do programa

### A Programação para WINDOWS

O WINDOWS é um sistema operacional de 32 bits com espaço de endereçamento de memória linear, não necessitando preocupação com o gerenciamento da memória. Também, os programas para WINDOWS não acessam diretamente o hardware dos dispositivos, podendo funcionar para qualquer placa de vídeo e qualquer impressora para a qual exista um controlador de dispositivo disponível para WINDOWS. Assim, o WINDOWS não é somente atraente em aparência, como também pode transmitir um grande número de informações ao usuário ao mesmo tempo.

No WINDOWS, tudo está interconectado. Se é desejado desenhar um gráfico na tela do vídeo, precisa do *“handle”* ou *“identificador do contexto do dispositivo”*. Para obter este identificador, precisa do *“identificador da janela”*. Para conseguir este último, precisa criar uma janela e estar preparado para receber *“mensagens”* para ela. Para receber e processar as mensagens, precisa de um *“procedimento de janela”*. Com isto, estrutura-se a codificação de um programa em WINDOWS.

Nesta estruturação, o programador gera a janela principal do programa, a qual pode conter chamadas à caixas de diálogos de entrada de dados, ou de saída de mensagens quaisquer, menus, botões, gerar gráficos, texto, utilizar o mouse, a impressora, gravar arquivos em disquetes ou no disco rígido, etc.

Deve-se levar em consideração que a programação para WINDOWS requer um conhecimento de uma pequena parte das principais funções e uma constante consulta à elas, seja na

forma impressa, seja em referências *on-line*. Isto, porque por mais experiente que o programador seja, não consegue memorizar as mais de 1000 funções que os aplicativos podem usar.

Toda função WINDOWS tem um nome descritivo com uma mistura de letras maiúsculas e minúsculas. O programador deve-se lembrar sempre que a linguagem C diferencia as letras minúsculas das letras maiúsculas. Logo, qualquer troca dessas, implica diretamente em um erro no programa.

Todas as funções do WINDOWS estão declaradas no arquivo WINDOWS.H, que é um arquivo de cabeçalho fornecido em qualquer ambiente de programação baseado em C que suporte o WINDOWS.

Um programa para WINDOWS se comunica com o WINDOWS por meio de um processo chamado “*ligação dinâmica*”. Assim, todo arquivo .EXE do WINDOWS contém referências às várias bibliotecas de ligações dinâmicas que ele utiliza e às funções ali contidas. Estas bibliotecas são arquivos .DLL, os quais, em sua maioria, estão localizados no diretório SYSTEM do WINDOWS. A falta de uma dessas bibliotecas, caso sejam utilizadas pelo programa, acarretam o seu não funcionamento.

### **As Mensagens WINDOWS**

O WINDOWS é uma interface gráfica com objetos chamados de janelas. Assim, toda a arquitetura de um programa WINDOWS é feita por mensagens que são enviadas do WINDOWS ao programa. Essas mensagens são chamadas que o WINDOWS faz às funções dentro do programa, cujos parâmetros descrevem a mensagem particular. Essa função no programa é denominada de “*procedimento de janela*”.

Toda janela de um programa WINDOWS tem um procedimento de janela associado. Esse procedimento é uma função que poderia estar dentro do programa ou na biblioteca de ligações dinâmicas. O WINDOWS envia uma mensagem para uma janela chamando o seu procedimento de janela, o qual executa uma tarefa se baseando na mensagem e em seguida retorna o controle ao WINDOWS. O procedimento de janela é a função que define o processamento de mensagens para a janela, para executar tarefas baseadas nas mensagens enviadas pelo WINDOWS.

Os programas WINDOWS contêm vários procedimentos de janelas, em que várias mensagens são declaradas para a comunicação com o WINDOWS. Na maioria das vezes, essas mensagens informam a janela sobre a entrada de dados pelo teclado ou pelo mouse (ou eventos). 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 redenhada.

### **Arquivos de um programa WINDOWS**

Na programação para WINDOWS, dependendo de que tipo de recursos o programa deverá conter, vários arquivos são criados para contruir o programa executável. O principal arquivo é o arquivo com extensão “.C”, que contém o código fonte do programa com todas as chamadas aos procedimentos de janelas, mensagens WINDOWS, funções, etc. Este arquivo pode conter funções criadas pelo programador construídas nele mesmo, ou pode conter chamadas a outras funções em outros arquivos com extensão “.C”.

Outros arquivos que dão base a um programa para WINDOWS são: o arquivo de cabeçalho, com extensão “.H” que normalmente contém informações sobre constantes utilizadas no programa, e o arquivo de recursos, com extensão “.RC” que contém o código de geração dos recursos do programa, isto é, dados do menu, botões, caixas de mensagens, ícones, figuras de bitmap, entre outros.

### **Escopo da Apostila**

Aqui é dada uma visão geral básica sobre a criação de programas para WINDOWS, desde a criação da janela, até a utilização da impressora. Todos os programas apresentados são construídos utilizando o compilador Borland C++, versão 5.00. Os arquivos de projeto para criar o programa executável (.EXE) são arquivos .IDE. Estes arquivos são criados pressionando o terceiro ícone da

barra de tarefas (New Project), onde aparece a janela Target Expert, onde as seguintes opções são selecionadas:

- Target Type: Application [.exe];
- Frameworks: Class Library;
- Platform: Win32;
- Target Model: GUI;
- Controls: BWCC;
- Libraries: Dynamic.

Como deseja-se criar programas em C, pressiona-se o botão AAdvanced, que chama a janela de opções avançadas, onde se seleciona o segundo radiobutton (c Node) no primeiro grupo, e a primeira caixa de seleção (.rc).

Com esse procedimento, é criado o arquivo .IDE, para criar os programas apresentados aqui. Porém, deve-se observar que nada impede que o programador utilize outro compilador, desde que os comandos sejam compatíveis.

## 2. CRIANDO A JANELA

Basicamente, quase todos os programas para WINDOWS criam uma janela principal, onde são apresentados todos os seus recursos gráficos e de texto, menus, botões, etc. Esta janela é construída através de um código básico, onde é definido o seu procedimento e contém as chamadas para as funções que o programa necessita, além das mensagens WINDOWS que determinam a comunicação do WINDOWS com o programa.

A janela principal básica contém apenas os botões do menu de sistema, podendo conter também os botões de maximização, minimização e de finalização e o nome do programa na barra de título. Para este caso, a janela principal pode ser redimensionada sempre que se desejar com a utilização do mouse.

O código básico para a criação da janela principal de um programa para WINDOWS é apresentado a seguir:

```
-----
                                SEDCURSO.C
-----

#include <windows.h>

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);

char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND      hjan;
    MSG        msg;
    WNDCLASS   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 =GetObject (WHITE_BRUSH);
    classejan.lpszMenuName =NULL;
    classejan.lpszClassName =szNomeAplic;

    if (!RegisterClass (&classejan)) return 0;

    hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
                        NULL, hCopia, NULL);
```

```

ShowWindow (hjan, iCmdMostrar);
UpdateWindow (hjan);

while (GetMessage (&msg, NULL, 0, 0)) {
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}

return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    switch (iMsg) {
        case WM_DESTROY:
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hjan, iMsg, wParam, lParam);
}

```

---

Este programa gera uma janela básica vista na Figura 1, a qual contém apenas a barra de título e os botões de menu de sistema, minimização, maximização e finalização, um ícone do sistema e um menu de sistema. A partir desta janela básica é que se constrói um programa específico para WINDOWS.

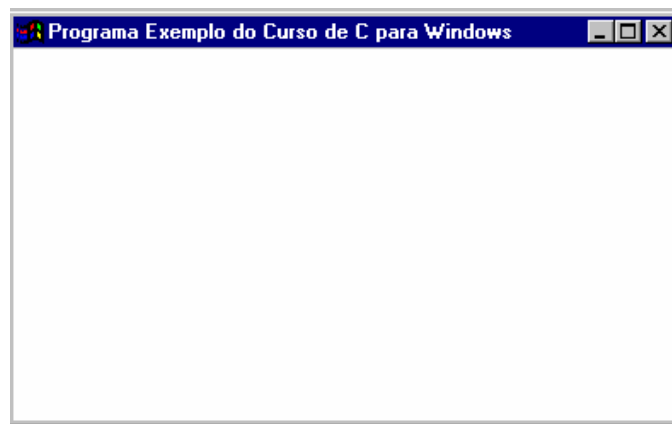


Figura 1

## O Arquivo do código fonte em C

O programa SEDCURSO.C tem duas funções: WinMain e ProcJan.

A função WinMain é o ponto de entrada do programa. Todo programa em C para WINDOWS possui esta função.

A função ProcJan é o procedimento da janela para o programa. Em todo programa WINDOWS, esta função possui o código para as chamadas às mensagens do WINDOWS e para processamento de eventos. Neste programa, somente uma mensagem é processada, como será visto adiante.

O procedimento de janela está presente em todo programa, seja ele uma janela principal de um aplicativo, ou de um botão criado dentro da janela principal.

Muito embora exista uma referência à função ProcJan em WinMain, todo procedimento de janela é chamado de dentro do WINDOWS, e não diretamente de algum lugar do código '.C'.

## SEDCURSO.C passo a passo

O código do programa SEDCURSO.C inicia com a declaração

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

para incluir o arquivo de cabeçalho WINDOW.H. Todo arquivo de cabeçalho tem a terminação “.H”, de header. São nos arquivos de cabeçalho que se encontram muitas declarações de constantes, funções utilizadas pelo programa principal, declarações de variáveis, estruturas. No caso do WINDOW.H, este é o arquivo que contém as declarações das funções do WINDOWS, as

estruturas, os novos tipos de dados e as constantes numéricas. Geralmente, além do WINDOWS.H, os programas WINDOWS podem apresentar outros arquivos de cabeçalho, onde se encontram as declarações do programa, tais como: constantes, funções, variáveis.

Embora todo programa WINDOWS tenha de apresentar obrigatoriamente a inclusão do WINDOWS.H, outros arquivos de cabeçalho não são necessariamente obrigatórios, podendo o programador declarar tudo no arquivo principal (.C). Naturalmente, uma vantagem de utilizar arquivos de cabeçalho é a redução do número de linhas no programa principal. De certa forma, isto facilita a avaliação e o entendimento do programa.

Após a declaração `#include <windows.h>`, é encontrada o protótipo da função `ProcJan`:

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

Toda função criada dentro de um programa WINDOWS deve ter um protótipo. Assim, existe a necessidade de declarar todos os protótipos das funções criadas.

As declarações:

```
char szTitulo[] = "Programa Exemplo do Curso de C para Windows";  
char szNomeAplic[] = "sedcurso";
```

servem para definir o nome para a janela, o qual é utilizado para ligar as declarações dos itens do programa à janela principal, tais como menu, botões, formato do mouse, bitmaps, ícones, janelas secundárias, teclas de atalho ou aceleradoras, entre outros, e o título da janela, como será visto adiante.

Após as declarações de variáveis globais e protótipos de funções é feita a entrada no programa através da função `WinMain`, que é sempre definida como:

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

Essa função usa uma seqüência de chamadas `WINAPI` e retorna um inteiro ao WINDOWS quando termina. Ela apresenta quatro parâmetros, que são:

- ✓ Parâmetro `hCopia`, que é o “identificador da cópia”. Isto é, o número que identifica o programa quando ele está em execução. Este parâmetro pode ser comparado a um identificador de tarefas, onde cada cópia do programa em execução obtém um número identificador para que o WINDOWS os diferencie;
- ✓ Parâmetro `hCopiaAnt`, que é um comando obsoleto. Nas versões anteriores ao WINDOWS 95, este parâmetro referenciava o identificador da ocorrência mais recente do programa que ainda estava ativa. Nas versões atuais do WINDOWS, este parâmetro é sempre nulo (NULL como declarado em WINDOWS.H).
- ✓ Parâmetro `szLinhaCmd` é um ponteiro para uma variável de texto (*string*) terminada por zero (finalizada pelo caractere ‘\0’), que contém os parâmetros da linha de comando passados ao programa. É possível executar um programa WINDOWS a partir do prompt do MS-DOS, ou digitando o nome do programa e o parâmetro na caixa de diálogo Executar, invocada a partir do menu Iniciar.
- ✓ Parâmetro `iCmdMostrar` é um número que indica como a janela deverá ser inicialmente exibida no WINDOWS, que pode ser um valor de 1 a 7 (valores declarados em WINDOWS.H por identificadores SW (ShowWindow) onde, `SW_SHOWNORMAL` = 1 - janela normal - e `SW_SHOWMINNOACTIVE` = 7 – janela minimizada).

Com a função `WinMain`, deve-se registrar a classe da janela. Uma janela é sempre criada com base em uma classe, a qual identifica o procedimento de janela que processa as mensagens. Mais de uma janela pode ser criada com base em uma única classe. Um exemplo disto é: todos os botões de janela do 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 mesma classe. Porém, ao criar uma janela, definem-se características adicionais específicas da janela.

O registro da classe da janela requer a utilização da função `RegisterClass`, a qual requer um único parâmetro que é um ponteiro para uma estrutura do tipo `WNDCLASS`. Esta estrutura também está definida nos arquivos de cabeçalho do WINDOWS.

Em primeiro lugar, declaram-se:

```

HWND      hjan;
MSG        msg;
WNDCLASS  classejan;

```

que são o identificador da janela, o identificador da estrutura de mensagens e a estrutura de WNDCLASS, respectivamente. A estrutura de WNDCLASS deve ser definida como a seguir:

```

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 = GetStockObject (WHITE_BRUSH);
classejan.lpszMenuName = NULL;
classejan.lpszClassName = szNomeAplic;

```

- O primeiro parâmetro da estrutura de WNDCLASS define o estilo da classe. Neste caso, há dois identificadores que são CS\_HREDRAW e CS\_VREDRAW, separados pelo operador *or* ( | ). Estes dois identificadores indicam que todas as janelas criadas com base nesta classe deverão ser completamente repintadas sempre que o tamanho horizontal da janela (CS\_HREDRAW) ou o tamanho vertical (CS\_VREDRAW) for alterado.
- O segundo parâmetro define que para esta janela o procedimento de janela é ProcJan. Este procedimento processará todas as mensagens para todas as janelas criadas com base nessa classe de janela.
- O terceiro e o quarto parâmetro definem uma reserva de espaço extra na estrutura de classe e na estrutura de janela que o WINDOWS mantém internamente. Esse espaço pode ser usado para um propósito específico do programa. Como este programa não utiliza esse recurso, os valores especificados são zero.
- O quinto parâmetro define apenas o identificador da cópia do programa.
- O sexto parâmetro define um ícone para todas as janelas criadas com base nessa classe de janela. O ícone é uma figura em bitmap que aparece na barra de tarefas do WINDOWS e na extremidade esquerda da barra de título dos programas. No caso aqui utilizado, IDI\_APPLICATION é o ícone predefinido pelo WINDOWS.
- O sétimo parâmetro carrega um cursor para o mouse. No caso deste programa, IDC\_ARROW é o cursor predefinido pelo WINDOWS: a seta.
- O oitavo parâmetro especifica a cor de fundo da área do cliente das janelas criadas com base nesta classe. hbrBackground significa identificador de um pincel, em que pincel é um termo gráfico que referencia um padrão colorido de pontos usados para preencher uma área. O WINDOWS tem diversos pincéis predefinidos, cujos identificadores podem ser obtidos utilizando a função GetStockObject. No caso do programa em análise, esta função retorna um identificador para um pincel branco (WHITE\_BRUSH).
- O nono parâmetro especifica o menu da classe da janela. Como este programa inicial não tem menu, o valor declarado é NULL.
- Por fim, o décimo parâmetro especifica o nome da classe da janela. Este nome está armazenado na variável szNomeAplic. Este nome pode ser o próprio nome do programa. No caso deste programa, o nome da classe da janela é sedcurso.

Após a declaração dos parâmetros da estrutura de WNDCLASS, registra-se a classe da janela chamando a função RegisterClass da seguinte forma:

```

if (!RegisterClass (&classejan)) return 0;

```

Com esta chamada, a janela é registrada, e assim pode-se criar a janela, a qual é feita pela chamada à função CreateWindow. Esta função requer que todas as informações sejam passadas como parâmetros, em vez de usar a estrutura de dados como faz RegisterClass:

```
hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
NULL, hCopia, NULL);
```

Os parâmetros da função `CreateWindow` são os seguintes:

`szNomeAplic` – nome da classe da janela. É por ele que a janela é associada à classe;

`szTitulo` – nome que aparecerá na barra de título da janela;

`WS_OVERLAPPEDWINDOW` – estilo normal da janela, cujo valor define que o programa deve ter o padrão básico: barra de título, caixa de menu do sistema à esquerda da barra de título, botões de maximização, minimização e de fechamento à direita da barra de título, e uma moldura de alteração de tamanho. Este é o valor padrão. Contudo, podem ser definidos valores próprios, para a janela só conter o que for desejado;

`CW_USEDEFAULT` – os quatro parâmetros que utilizam este valor, indicam a posição inicial, a posição final, a largura e a altura da janela, respectivamente. O valor dado neste programa determina que o `WINDOWS` decide aleatoriamente as posições e o tamanho da janela;

`HWND_DESKTOP` – identificador da janela mãe a que pertence a janela que está sendo criada. Esse valor definido determina que a janela pertence ao *desktop* do `WINDOWS`. Este parâmetro pode ser definido apenas como `NULL`, identificando o valor padrão, sendo igual a `HWND_DESKTOP`;

`NULL` – identificador do menu da janela é definido com este valor desde que o programa não tem um menu;

`hCopia` – identificador da cópia do programa. Está associado ao identificador da cópia passado ao programa como um parâmetro de `WinMain`.

`NULL` – parâmetro de criação. Esse parâmetro é definido como um ponteiro para a janela criada através da estrutura `CREATESTRUCT`. Esse ponteiro é referenciado ao parâmetro `lParam` da mensagem `WM_CREATE`.

Com a chamada à `CreateWindow`, exibe-se a janela através das funções:

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

A função `ShowWindow` tem dois parâmetros: o identificador da janela criada por `CreateWindow` e o valor `iCmdMostrar` que é passado à `WinMain`. Este segundo parâmetro define como a janela será exibida inicialmente: normalmente (`SW_SHOWNORMAL`) ou minimizada na barra de tarefas (`SW_SHOWMINNOACTIVE`).

A segunda função faz com que a área do cliente seja pintada.

Tendo criado a janela, o procedimento de janela deve ser ativado para que o programa possa receber as entradas do usuário através do teclado ou mouse. Isto é feito através do laço de mensagens a seguir:

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

O laço de mensagens é criado para que o `WINDOWS` possa traduzir os eventos de entrada do programa, como pressionamentos dos botões do mouse ou do teclado. Para cada programa em execução, o `WINDOWS` mantém uma fila de mensagens. Cada evento de entrada, o `WINDOWS` o traduz para uma mensagem, colocando-a na fila de mensagens para ser executada. Observe que o laço é feito pela função `while {...}`. O parâmetro de teste do laço de mensagens é a chamada à função `GetMessage`, que retira uma mensagem da fila. A função `GetMessage` utiliza quatro parâmetros: ponteiro para o `WINDOWS` da variável `msg` (que é uma estrutura) e três parâmetros que estão definidos como `NULL`, `0` e `0`, indicando que são retornadas todas as mensagens de todas as janelas criadas pelo programa.

Todas as mensagens têm um identificador no arquivo de cabeçalho `WINDOWS.H` que inicia com o prefixo `WM_`, de `Windows Message`.



As declarações internas ao laço determinam, respectivamente: passagem da estrutura msg de volta ao WINDOWS para tradução de algum evento e passagem da estrutura msg de volta ao WINDOWS para transmissão da mensagem ao procedimento de janela apropriado, para seu processamento (WINDOWS chama o procedimento da janela).

Por fim, o programa retorna o parâmetro wParam da mensagem ao WINDOWS:

```
return msg.wParam;
```

O procedimento de janela sempre é definido como:

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

cujos parâmetros são:

- ✓ hjan – identificador para a janela que está recebendo a mensagem;
- ✓ iMsg – número que identifica a mensagem;
- ✓ wParam – fornece uma informação sobre a mensagem;
- ✓ lParam – fornece outra informação sobre a mensagem.

Os dois últimos parâmetros são conhecidos como parâmetros da mensagem, e contém informações específicas para cada tipo de mensagem. Por exemplo: as posições *x* e *y* do mouse podem ser informações contidas nesses dois parâmetros.

O procedimento de janela recebe mensagens que são identificadas por um número que é um parâmetro para iMsg. Para ProcJan processar uma mensagem recebida, precisa determinar que mensagem foi recebida. Geralmente isto é feito através da função switch e case. Assim, a função switch avalia iMsg e, dependendo de qual identificador ela contém (detectado pela função case), processa a devida mensagem. No programa SEDCURSO.C, o procedimento de janela é o seguinte:

```
switch (iMsg) {  
    case WM_DESTROY:  
        PostQuitMessage (0);  
        return 0;  
}
```

Vê-se que o único identificador de mensagem que o procedimento de janela processa é o que define o término do programa WM\_DESTROY: pressão sobre o botão de término do programa localizado ao lado direito da barra de título (×); pressionamento das teclas ALT+F4 conjuntamente; duplo pressionamento sobre o menu do sistema ao lado esquerdo da barra de título onde se encontra o ícone do programa ou abrindo o menu do sistema e selecionando Fechar.

Finalmente, todas as mensagens que o procedimento de janela não processar precisam ser passadas à função DefWindowProc da seguinte forma:

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

O valor devolvido por essa função precisa ser retornado dentro do procedimento de janela. É essencial chamar a função para efetuar o processamento padrão de todas as mensagens que o procedimento de janela não processar.

### 3. INTRODUZINDO MENU E BOTÕES

Tendo a janela principal, é necessário implementar controles para o programa. Geralmente, esses controles são apresentados em um menu. Os menus podem ser simples, onde um pressionamento do botão do mouse sobre um item do menu gera um comando para o programa; comum, em que um pressionamento do botão do mouse sobre um item do menu abre uma lista de subitens, e ainda com submenus, em que um item do menu abre uma lista de itens, onde alguns ou todos os itens dessa lista levam à sub-menus, e assim por diante. Geralmente os menus mais utilizados são apresentados como uma lista de itens abaixo da barra de título. Porém, pode-se criar menus com botões na mesma posição do menu principal, ou mesmo menus de botões flutuantes, que podem ser arrastados para qualquer posição da área de trabalho do aplicativo.

Quando se utiliza um menu com botões, necessita-se saber o que cada botão realiza. Assim, criam-se os *Tooltips*, que são pequenas janelas informativas que aparecem em um determinado lugar em que o mouse se posiciona.

Nos menus, normalmente é utilizado o teclado para sua abertura através de teclas de atalho. Nesse caso, a tecla Alt mais alguma outra tecla abre um item do menu do programa. Esta outra tecla a ser pressionada com a tecla Alt, sempre é representada no item do menu por uma letra sublinhada.

Uma outra forma de lidar com menus é através das teclas aceleradoras. Essas teclas podem ser definidas para não ser necessário sempre a utilização do mouse ou a tecla Alt. Geralmente, a utilização da tecla Ctrl com alguma outra tecla define um comando direto ao programa. Porém, não necessariamente deve-se utilizar esta tecla. Exemplo disso é a utilização da tecla F1 para chamar a ajuda em muitos programas.

A seguir é apresentado o código do programa SEDCURSO.C incrementado das linhas de código para a criação do menu, botões de menu, *Tooltips* e teclas de atalho e teclas aceleradoras na janela principal, em conjunto com os arquivos complementares: SEDCURSO.H e SEDCURSO.RC. As linhas modificadas ou introduzidas são apresentadas em texto não itálico com um caractere \* ao lado direito.

```
-----
                                SEDCURSO.C
-----

#include <windows.h>
#include <commctrl.h>
#include "sedcurso.h"

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
void InitToolbar (void);

TBBUTTON    tbButtons [NUMBUTTONS];
HWND        tbwnd;
int         ToolBarActive;
UINT        wSelection = IDM_E1;
HMENU       hMenu;

char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND        hjan;
    MSG         msg;
    WNDCLASS    classejan;
    HACCEL      hAccel;

    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 =GetStockObject (WHITE_BRUSH);
    classejan.lpszMenuName  =szNomeAplic;
    classejan.lpszClassName =szNomeAplic;

    if (!RegisterClass (&classejan)) return 0;

    hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
                        NULL, hCopia, NULL);

    hAccel = LoadAccelerators (hCopia, szNomeAplic);
    InitToolbar ();
    InitCommonControls();

    tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
                            IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
                            16, 16, sizeof(TBBUTTON));

    ShowWindow (hjan, iCmdMostrar);
}
```

```

UpdateWindow (hjan);

while (GetMessage (&msg, NULL, 0, 0)) {
    if (!TranslateAccelerator(hjan, hAccel, &msg)){
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}

return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    LPTOOLTIPTTEXT TTtext;
    switch (iMsg) {
        case WM_NOTIFY:
            TTtext = (LPTOOLTIPTTEXT) lParam;
            if (TTtext->hdr.code == TTN_NEEDTEXT)
                switch (TTtext->hdr.idFrom){
                    case IDM_GE: TTtext->lpszText = "Girar Esteira";
                        break;
                    case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
                        break;
                    case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
                        break;
                    case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
                        break;
                    case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
                        break;
                }
            return 0;

        case WM_COMMAND:
            switch (LOWORD(wParam)) {
                case IDM_ABRIR:
                    MessageBox(hjan, "Não pode abrir o Arquivo", szNomeAplic, MB_OK);
                    break;
                case IDM_SALVAR:
                    MessageBox(hjan, "Não pode salvar o Arquivo", szNomeAplic, MB_OK);
                    break;
                case IDM_IMPRIMIR:
                    MessageBox (hjan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
                    break;
                case IDM_GE:
                    break;
                case IDM_PP:
                    break;
                case IDM_GB:
                    break;
                case IDM_SB:
                    break;
                case IDM_PT:
                    break;
                case IDM_E1:
                case IDM_E2:
                case IDM_E3:
                    hMenu = GetMenu(hjan);
                    CheckMenuItem (hMenu, wSelection, MF_UNCHECKED);
                    wSelection = LOWORD(wParam);
                    CheckMenuItem (hMenu, wSelection, MF_CHECKED);
                    break;
                case IDM_SAIR:
                    if (IDOK == MessageBox (hjan, "Você deseja realmente terminar o programa?",
                        szNomeAplic, MB_OKCANCEL | MB_ICONEXCLAMATION))
                        SendMessage (hjan, WM_CLOSE, 0, 0L);
                    break;
                case IDM_MOSTRAR:
                    ToolBarActive = 1;
                    ShowWindow(tbwnd, SW_RESTORE);
                    InvalidateRect(hjan, NULL, 1);
            }
    }
}

```

```

        break;
    case IDM_ESCB:
        ToolBarActive = 0;
        ShowWindow(tbwnd, SW_HIDE);
        InvalidateRect(hjan, NULL, 1);
        break;

    case IDM_SOBRE:
        MessageBox(hjan, "tPrograma de Simulação de Sistema\r\npara Aprendizado de
        Programação\r\nem C para WINDOWS!\r\n\r\nEduard Montgomery Meira Costa,
        DSc\r\n\r\n\t\t\t2002", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        return 0;
    }
    return 0;

case WM_CLOSE:
    DestroyWindow (hjan);
    return 0;

case WM_DESTROY:
    PostQuitMessage (0);
    return 0;
}
return DefWindowProc (hjan, iMsg, wParam, lParam);
}

```

```

void InitToolbar () {
tbButtons[0].iBitmap = 0;
tbButtons[0].idCommand = IDM_GE;
tbButtons[0].fsState = TBSTATE_ENABLED;
tbButtons[0].fsStyle = TBSTYLE_BUTTON;
tbButtons[0].dwData = 0l;
tbButtons[0].iString = 0;

tbButtons[1].iBitmap = 0;
tbButtons[1].idCommand = 0;
tbButtons[1].fsState = TBSTATE_ENABLED;
tbButtons[1].fsStyle = TBSTYLE_SEP;
tbButtons[1].dwData = 0l;
tbButtons[1].iString = 0;

tbButtons[2].iBitmap = 1;
tbButtons[2].idCommand = IDM_PP;
tbButtons[2].fsState = TBSTATE_ENABLED;
tbButtons[2].fsStyle = TBSTYLE_BUTTON;
tbButtons[2].dwData = 0l;
tbButtons[2].iString = 0;

tbButtons[3].iBitmap = 2;
tbButtons[3].idCommand = IDM_GB;
tbButtons[3].fsState = TBSTATE_ENABLED;
tbButtons[3].fsStyle = TBSTYLE_BUTTON;
tbButtons[3].dwData = 0l;
tbButtons[3].iString = 0;

tbButtons[4].iBitmap = 3;
tbButtons[4].idCommand = IDM_SB;
tbButtons[4].fsState = TBSTATE_ENABLED;
tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0l;
tbButtons[4].iString = 0;

tbButtons[5].iBitmap = 0;
tbButtons[5].idCommand = 0;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0l;
tbButtons[5].iString = 0;

tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0l;
}

```

```

tbButtons[6].iString = 0;
}
-----
                                SEDCURSO.H
-----
#define IDM_ABRIR                0
#define IDM_SALVAR                10
#define IDM_IMPRIMIR             20
#define IDM_SAIR                 30
#define IDM_GE                   40
#define IDM_PP                   50
#define IDM_GB                   60
#define IDM_SB                   70
#define IDM_PT                   80
#define IDM_E1                   90
#define IDM_E2                   100
#define IDM_E3                   110
#define IDM_MOSTRAR               120
#define IDM_ESCB                 130
#define IDM_SOBRE                 140

#define IDTB_BMP                 300

#define IDM_TOOLBAR               200

#define NUMBUTTONS               7
-----
                                SEDCURSO.RC
-----
#include "sedcurso.h"
IDTB_BMP BITMAP "toolbarsedcurso.bmp"
sedcurso MENU {
    POPUP "&Arquivo" {
        MENUITEM "&Abrir\t^A", IDM_ABRIR
        MENUITEM "&Salvar\t^S", IDM_SALVAR
        MENUITEM "&Imprimir\t^I", IDM_IMPRIMIR
        MENUITEM SEPARATOR
        MENUITEM "Sai\r\t^R", IDM_SAIR
    }
    POPUP "&Sistema" {
        MENUITEM "&Girar Esteira\t^G", IDM_GE
        POPUP "&Braço Robótico" {
            MENUITEM "&Pegar Peça\t^P", IDM_PP
            MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
            MENUITEM "S&oltar Peça\t^O", IDM_SB
        }
        MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
    }
    POPUP "&Cores" {
        MENUITEM "Esquema &1\t^H", IDM_E1
        MENUITEM "Esquema &2\t^J", IDM_E2
        MENUITEM "Esquema &3\t^K", IDM_E3
    }
    POPUP "&Botões" {
        MENUITEM "&Mostrar\t^M", IDM_MOSTRAR
        MENUITEM SEPARATOR
        MENUITEM "Es&conder\t^C", IDM_ESCB
    }
    POPUP "&Ajuda" {
        MENUITEM "So&bre\t^F1", IDM_SOBRE
    }
}

sedcurso ACCELERATORS {
    "^A", IDM_ABRIR
    "^S", IDM_SALVAR
    "I", IDM_IMPRIMIR, CONTROL
    "^R", IDM_SAIR
}

```

```

"AG",    IDM_GE
"AP",    IDM_PP
"AB",    IDM_GB
"AO",    IDM_SB
"AE",    IDM_PT
"H",     IDM_E1, CONTROL
"J",     IDM_E2
"K",     IDM_E3
"M",     IDM_MOSTRARB
"C",     IDM_ESCB
VK_F1,   IDM_SOBRE, VIRTKEY
}

```

A inclusão/modificação do código no arquivo SEDCURSO.C e os arquivos SEDCURSO.H e SEDCURSO.RC geram a janela vista na Figura 2, incluindo um menu com um item apresentando um submenu, além de botões que realizam parte dos comandos do menu, os quais podem ser removidos ou mostrados, bem como teclas aceleradoras para os comandos do menu.

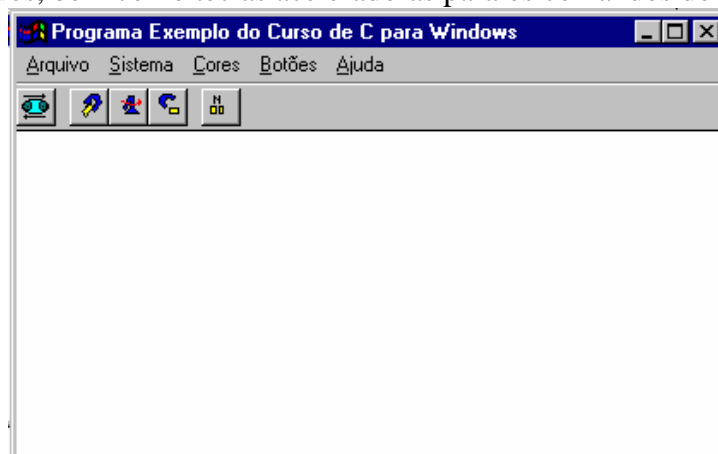


Figura 2

### Análise do Código Incluído no Arquivo SEDCURSO.C

Inicialmente, como neste programa são incluídos botões de menu, a primeira linha de código a ser incluída é:

```
#include <commctrl.h>
```

e como utilizam-se recursos adicionais (menus e teclas aceleradoras) inclui-se a linha:

```
#include "sedcurso.h"
```

Esses arquivos de cabeçalho são necessários, desde que o arquivo COMMCTRL.H é onde se encontram as definições para os controles comuns do WINDOWS, que aqui são os *Tooltips*, e o arquivo SEDCURSO.H que é o arquivo de cabeçalho do programa MENU: onde estão as definições dos números de identificação (ID...) das constantes utilizadas no programa. Os números de identificação são declarados dessa maneira, com “...” sendo uma letra maiúscula de identificação do recurso: M para menus, C para cursores, I para ícones, TB para toolbar (barra de ferramentas) e outros.

Após a inclusão dos arquivos de cabeçalho, encontra-se a declaração:

```
void InitToolbar (void);
```

que é o protótipo da função que cria os botões de menu.

As seguintes variáveis globais são incluídas neste programa:

```

TBBUTTON  tbButtons [NUMBUTTONS];
HWND      tbwnd;
Int        ToolBarActive;
UINT       wSelection = IDM_E1;
HMENU      hMenu;

```

Estas variáveis são definidas, respectivamente, para: definição dos botões da barra de ferramenta (botões de menu) – tipo TBBUTTON; identificador para a barra de ferramentas que é uma janela (tipo HWND); um inteiro (tipo int) cujo valor determina a apresentação (valor 1) ou a

remoção (valor 0) da barra de ferramentas; um inteiro (tipo `UINT`) para guardar itens selecionados no menu e um identificador para o menu (tipo `HMENU`).

Neste programa, uma outra linha de código incluída, encontra-se na função `WinMain`, que é:

```
HACCEL hAccel;
```

que define um identificador para as teclas aceleradoras que serão utilizadas no programa. Também, na definição da classe da janela, tem-se:

```
classejan.lpszMenuName = szNomeAplic;
```

Esta declaração determina um menu que será carregado e que é identificado no arquivo de recursos (.RC) pelo nome definido na variável `szNomeAplic`, isto é, `sedcurso`.

O carregamento das teclas aceleradoras é feito através da linha de código:

```
hAccel = LoadAccelerators(hCopia, szNomeAplic);
```

A função `LoadAccelerators` carrega um valor identificador para a variável `hAccel`, utilizando como parâmetros o identificador da cópia do programa (`hCopia`) e o nome da classe da janela (`szNomeAplic`).

Para serem criados os botões da barra de ferramentas, as funções:

```
InitToolbar();  
InitCommonControls();
```

são chamadas, que definem as figuras de cada botão em referência ao bitmap que as contém (esta função será explicada mais detalhadamente, adiante), e vincula a biblioteca dinâmica (DLL) dos controles comuns e inicializa o subsistema de controles comuns do `WINDOWS`, para os *Tooltips* poderem ser utilizados, respectivamente. Após a inicialização dos botões e dos controles comuns do `WINDOWS`, a barra de ferramentas é criada através da declaração:

```
tbwnd = CreateToolbarEx(hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,  
IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,  
0, 16, 16, sizeof(TBBUTTON)); *
```

Esta declaração cria um identificador para a barra de ferramentas. A função `CreateToolbarEx` tem como parâmetros:

- 1) o identificador da janela principal (`hjan`);
- 2) o estilo da barra de ferramentas (`WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS`), que define que inicialmente os botões devem estar visíveis, é uma janela filha (parâmetro obrigatório), com borda e com utilização de *Tooltips*;
- 3) o identificador da barra de ferramentas (`IDM_TOOLBAR`);
- 4) o número de bitmaps que formarão os ícones da barra de ferramentas, incluindo os espaços separadores (`NUMBUTTONS`);
- 5) o identificador da cópia do programa (`hCopia`);
- 6) o identificador do bitmap que contém os ícones da barra de ferramentas (`IDTB_BMP`);
- 7) o identificador da estrutura que contém informações relativas a cada botão (`tbButtons`);
- 8) o número de botões, incluindo os espaços separadores da barra de ferramentas (`NUMBUTTONS`);
- 9) e 10) a largura e a altura dos botões (definidos como 0, as dimensões dos botões são fornecidas automaticamente);
- 11) e 12) a largura e a altura das imagens dentro de cada botão e,
- 13) o tamanho da estrutura `TBBUTTON` que contém informações relativas a cada botão (`sizeof(TBBUTTON)`).

Ainda em `WinMain`, para que as teclas aceleradoras sejam sempre avaliadas, a declaração

```
if (!TranslateAccelerator(hjan, hAccel, &msg)){  
    TranslateMessage(&msg);  
    DispatchMessage(&msg);  
}
```

é incluída no laço de mensagens. Assim, se uma tecla aceleradora é pressionada, o valor retornado pela função `TranslateAccelerator` é verdadeiro (`TRUE`). Caso contrário, é falso (`FALSE`).

No caso do procedimento de janela (`ProcJan`), inicialmente, tem-se a definição da variável estrutura dos *Tooltips*:

```
LPTOOLTIPTEXT TTtext;
```

que é necessária à criação dos *Tooltips*.

Para a avaliação das mensagens, são incluídas as que dizem respeito à apresentação dos *Tooltips* e à avaliação dos itens do menu:

**WM\_NOTIFY:** - mensagem utilizada para o **WINDOWS** gerar as janelas de texto informativo quando o mouse repousa sobre um botão de menu. A variável **TTtext** é definida sobre a estrutura **LPTOOLTIPTTEXT** apontada por **lParam**. Se uma *Tooltip* for requisitada, definida pela função **if (...)**, então a variável **TTtext** que aponta para **hdr.code** (variável da estrutura **LPTOOLTIPTTEXT**) recebe a notificação de que existe uma janela de dica a ser apresentada (**TTN\_NEEDTEXT**). Dessa forma, com a declaração

```
switch (TTtext->hdr.idFrom){...}
```

avalia-se o valor de **TTtext**, em que **hdr.idFrom**, contém o valor (ID) do botão para o qual a janela de dica é necessária. As definições das dicas a serem apresentadas são geradas pela avaliação (função **case**) dos identificadores de menu (**IDM\_...**), determinando o texto (**TTtext->lpszText**) a ser apresentado na janela de dica:

```
case IDM_GE: TTtext->lpszText = "Girar Esteira";
break;
case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
break;
```

...

Por fim, finaliza-se a mensagem **WINDOWS** com

```
return 0;.
```

**WM\_COMMAND:** - mensagem **WINDOWS** utilizada para processar a seleção de menu. Quando essa mensagem é recebida, o valor de **LOWORD(wParam)** contém o valor do ID do item do menu selecionado. Através da função **switch**, pode-se avaliar neste parâmetro qual item foi selecionado e processar o procedimento a que o item se refere. Geralmente, ao final de cada processamento de um item de menu, deve-se finalizar com **break;** para terminar a função **switch... case**. Quando vários itens de menu podem ser processados pela mesma função, a finalização pode ser feita como no caso desse programa:

```
case IDM_E1:
case IDM_E2:
case IDM_E3:
    hMenu = GetMenu(hjan);
    CheckMenuItem (hMenu, wSelection, MF_UNCHECKED);
    wSelection = LOWORD(wParam);
    CheckMenuItem (hMenu, wSelection, MF_CHECKED);
    break;
```

Assim, ao selecionar qualquer um desses itens do menu, a variável **hMenu** tem seu novo valor determinado, onde se há um item selecionado é eliminada a sua seleção através da declaração

```
CheckMenuItem (hMenu, wSelection, MF_UNCHECKED);
```

a variável **wSelection** obtém o valor do novo item selecionado e este item é, então, selecionado pela mesma função **CheckMenuItem**, com o último parâmetro definido como **MF\_CHECKED**. Quando se seleciona os itens do menu (**IDM\_ABRIR**, **IDM\_SALVAR** e **IDM\_IMPRIMIR**), é criada uma caixa de diálogos através da função:

```
MessageBox(hjan, "mensagem", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
```

Essa função cria uma caixa de diálogo utilizada para enviar mensagens, onde seus parâmetros são: 1) **hjan** – identificador do procedimento de janela; 2) a mensagem a ser apresentada; 3) o nome da janela definido na variável **szNomeAplic** e 4) os “*flags*” da caixa de diálogo separados pelo operador *or* ( **|** ). Para essas mensagens, as duas primeiras apresentam apenas um botão de confirmação (**MB\_OK**) e a última inclui também um ícone de exclamação aparece em seu lado esquerdo (**MB\_ICONEXCLAMATION**). Os outros itens do menu são: **IDM\_GE**, **IDM\_GB**, **IDM\_PT**, **IDM\_PP** e **IDM\_SB** que não processam nada neste programa (mas serão utilizados adiante para gerar animação gráfica); **IDM\_SOBRE**, que apresenta uma caixa de diálogo dando informações sobre o programa; **IDM\_SAIR**, que gera uma caixa de diálogo perguntando se realmente é desejado terminar o programa. Nesta caixa de diálogo, dois botões são apresentados (**OK** e **CANCEL** definidos por **MB\_OKCANCEL**) e um ícone de exclamação (**MB\_ICONEXCLAMATION**). Se for pressionado o botão **OK**, a função **MessageBox** retorna o valor **IDOK**, satisfazendo o **if**, que processa a função

```
SendMessage (hjan, WM_CLOSE, 0, 0L);
```



Esta função é utilizada para enviar mensagens específicas para a janela ou para o WINDOWS. Nesse caso, a mensagem é enviada para a janela, a qual é de término do programa (WM\_CLOSE) que finaliza o programa (vista adiante). Por fim, caso o item selecionado do menu seja IDM\_MOSTRARB, a variável `ToolBarActive`, torna-se 1, determinando que os botões devem ser mostrados

```
ShowWindow(tbwnd, SW_RESTORE);
```

em que `SW_RESTORE` é o valor de `ShowWindow` que determina que a barra de ferramentas (tbwnd) deve ser apresentada em seu tamanho normal; e caso o item selecionado do menu seja `IDM_ESCB`, os botões não são apresentados na janela. Em ambos os casos, apresenta-se a função

```
InvalidateRect(hjan, NULL, 1)
```

que força uma chamada a `WM_PAINT`, invalidando a “*área do cliente*” (área da janela utilizada para processamento do programa: apresentação de texto e gráficos). O primeiro parâmetro desta função referencia à janela a ser repintada; o segundo referencia às coordenadas do retângulo a ser repintado na janela e, o terceiro parâmetro define se esta área do retângulo é apagado (valor `TRUE` ou 1) ou não (valor `FALSE` ou 0). No caso do segundo parâmetro, sua definição como `NULL` define que toda a “*área do cliente*” deve ser repintada. A mensagem `WM_PAINT` será melhor analisada mais adiante.

Deve-se observar que ao final de toda função `case` para uma mensagem `WINDOWS`, é necessário terminá-la com a declaração `break`; se a mensagem necessita ser processada por `DefWindowProc`, ou `return 0`; caso contrário.

**WM\_CLOSE:** - mensagem utilizada para fechar o programa a partir da seleção do menu. A função

```
DestroyWindow (hjan);
```

termina a execução do programa. Deve-se observar que a mensagem `WM_DESTROY`, não necessariamente deve ser declarada.

Em `MENU.C`, também é incluída a função de inicialização da barra de tarefas:

```
void InitToolBar ()
```

a qual tem como objetivo definir quais os botões da barra de tarefas, em que seqüência os botões devem aparecer, e espaços separadores, se houverem. Esta função se apresenta como uma série dados para a estrutura `tbButtons`, em que cada índice da estrutura contém as informações sobre uma posição: botão (número identificador do bitmap, identificador do comando do menu – `ID..._`, estado do botão, estilo do botão, dados definidos pelo usuário e índice de um caractere opcional associado ao botão) ou espaço separador.

Para um botão, a função deve se apresentar da seguinte maneira:

```
tbButtons[0].iBitmap = 0;
tbButtons[0].idCommand = IDM_GE;
tbButtons[0].fsState = TBSTATE_ENABLED;
tbButtons[0].fsStyle = TBSTYLE_BUTTON;
tbButtons[0].dwData = 0;
tbButtons[0].iString = 0;
```

Observe no programa que o identificador do bitmap (primeiro parâmetro) define qual bitmap deve aparecer no botão, o qual deve estar relacionado com o respectivo identificador de comando do menu (no segundo parâmetro). O terceiro parâmetro deve ser colocado como `TBSTATE_ENABLED`, para definir um botão habilitado. O quarto parâmetro deve ser declarado como `TBSTYLE_BUTTON`, determinando o estilo do controle como um botão.

Para um espaço separador, inclui-se entre as definições dos botões na respectiva posição o seguinte código:

```
tbButtons[1].iBitmap = 0;
tbButtons[1].idCommand = 0;
tbButtons[1].fsState = TBSTATE_ENABLED;
tbButtons[1].fsStyle = TBSTYLE_SEP;
tbButtons[1].dwData = 0;
tbButtons[1].iString = 0;
```

Assim, não há identificação de bitmaps, nem identificadores de comandos. Também, o estilo do botão é definido como `TBSTYLE_SEP`, que informa que é um espaço separador de botões.

Deve-se ver que esta função contém um total de identificações igual a NUMBUTTONS, que incluem os dez botões e os dois espaços separadores.

### O Arquivo SEDCURSO.H

Este arquivo contém as informações sobre as definições de todos os identificadores usados no programa. Inicialmente, apresentam-se os identificadores dos itens do menu:

```
#define IDM_ABRIR      0
#define IDM_SALVAR     10
#define IDM_IMPRIMIR   20
#define IDM_SAIR       30
#define IDM_GE         40
#define IDM_PP         50
#define IDM_GB         60
#define IDM_SB         70
#define IDM_PT         80
#define IDM_E1         90
#define IDM_E2        100
#define IDM_E3        110
#define IDM_MOSTRAR    120
#define IDM_ESCB       130
#define IDM SOBRE     140
```

os identificadores do bitmap utilizado para criar os botões:

```
#define IDTB_BMP      300
```

o identificador da barra de tarefas (barra dos botões):

```
#define IDM_TOOLBAR   200
```

e o número de botões:

```
#define NUMBUTTONS    7
```

incluindo os espaços separadores a serem apresentados na janela. Nesse caso, há cinco botões e dois espaços de separação, como pode ser visto na Figura 2.

Observe que alguns identificadores devem ter valores exatos em relação à sua utilização no programa. Os outros (como identificadores de menu), se não tiverem uma ligação no programa principal que exijam sua definição com um valor específico (caso de utilização para definição de índices de vetores de recursos do tipo pincéis, etc.) podem ter valores quaisquer, contudo que não apresentem valores iguais.

### O Arquivo SEDCURSO.RC

Este é o arquivo de recursos do programa. Como ele depende das definições dos identificadores definidos no arquivo SEDCURSO.H, este arquivo deve inicialmente ser incluído através da declaração:

```
#include "sedcurso.h"
```

Para o caso aqui estudado, a declaração do bitmap que contém as figuras dos botões é declarado em consonância com o identificador de bitmap declarado no arquivo SEDCURSO.H:

```
IDTB_BMP BITMAP "toolbarsedcurso.bmp"
```

O bitmap utilizado para as figuras dos botões é mostrado na Figura 3, e tem como título toolbarsedcurso.bmp.

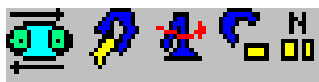


Figura 3

Este bitmap pode ser criado utilizando um programa gráfico qualquer (como o PAINT) que tenha a seleção de gravação em .BMP.

Criado o bitmap, inicia-se o recurso do menu. Inicialmente, apresenta-se a declaração do menu, da forma: “Nome do Aplicativo” definido em szNomeAplic, seguido da definição MENU, indicando que o código é para a construção do menu. No caso desse programa, o nome do aplicativo é Menu, e a declaração para sua construção:

```
sedcurso MENU {...}
```

Dentro das chaves, o código é apresentado com a declaração:

```
POPUP "Item" {...}
```

que indica que há um submenu, ou uma lista de itens que é aberta, quando o Item de POPUP é selecionado. Essa lista de itens está definida dentro das chaves como sub-itens (que realizam as funções do programa) declarados por MENUITEM. Assim, a declaração:

```
POPUP "&Sistema" {  
    MENUITEM "&Girar Esteira\t^G", IDM_GE  
    POPUP "&Braço Robótico" {  
        MENUITEM "&Pegar Peça\t^P", IDM_PP  
        MENUITEM "Girar &Braço Robótico\t^B", IDM_GB  
        MENUITEM "S&oltar Peça\t^O", IDM_SB  
    }  
    MENUITEM "P&eças Trabalhadas\t^E", IDM_PT  
}
```

cria um item principal definido por Sistema que, ao ser selecionado abre a lista de itens: Girar Esteira, Braço Robótico e Peças Trabalhadas. O item Braço Robótico é definido como um Item de POPUP, o que abre um outro submenu que contém a lista de itens: Pegar peça, Girar Braço Robótico e Soltar Peça. Cada declaração MENUITEM define uma seleção final de um item, que processa uma função no programa. Todo Item de POPUP pode ter várias chamadas à sub-menus (outros Itens de POPUP).

Observa-se que cada item do menu é definido por uma palavra, em que aparece o símbolo & em uma determinada posição. Este símbolo define que a letra posterior aparece sublinhada, sendo uma tecla de atalho. Por exemplo, na declaração

```
MENUITEM "&Braço Robótico\t^G", IDM_GE
```

quando o item do menu Sistema é selecionado (observe que o S está sublinhado) aparece na lista de itens o item 'Braço Robótico ^G'. Assim, a letra B que está sublinhada define que se for pressionada a tecla 'b' no teclado, é o mesmo que selecionar esse item com o mouse. Por outro lado, se nenhuma lista referente a um Item de POPUP está aberto, o menu pode ser inicializado pela tecla Alt, e o pressionamento da letra sublinhada de um dos Itens de POPUP, determina que este item está sendo selecionado.

Também se deve observar que o '\t' antes de '^'(letra)' define uma tabulação para afastar a palavra que determina o nome do item da declaração '^'(letra)' que é a definição da tecla aceleradora, onde o acento circunflexo '^' define a tecla Ctrl. Esta forma só é utilizada se o programa apresentar teclas aceleradoras.

Para cada declaração de MENUITEM, deve-se associar o número do identificador do menu (IDM\_...), para que ele seja avaliado no procedimento de janela. Caso exista um item de menu que não abre nenhuma lista de itens, um pressionamento do botão do mouse sobre este item, ou a utilização da tecla aceleradora, ou mesmo a utilização da tecla de atalho determina a execução direta da função associada com o item. Um exemplo disto é a declaração

```
MENUITEM "&Sair -> ^S", IDM_SAIR
```

que determina a saída do programa MENU.

A declaração

```
MENUITEM SEPARATOR
```

coloca no menu uma barra de separação entre os itens.

A declaração das teclas aceleradoras também é colocada no arquivo de recursos. Esta é feita igualmente à declaração do menu, isto é, nome do aplicativo definido em szNomeAplic seguido da declaração ACCELERATORS. Dentro desta função são encontradas todas as definições das teclas aceleradoras de menu, que são declaradas como:

Key, IDM\_..., Type Option

em que Key indica a tecla que seleciona o item; IDM\_... indica o valor do número identificador do menu; Type especifica se a tecla Key é comum (padrão) ou uma tecla virtual (discutida adiante) e Option pode ser uma das seguintes macros: NOINVERT, ALT, SHIFT e CONTROL. NOINVERT evita que o item de menu selecionado seja colocado em destaque quando sua tecla aceleradora é pressionada; ALT especifica a tecla ALT; SHIFT especifica a tecla SHIFT e CONTROL especifica a tecla Ctrl.

O valor de Key será um caractere entre aspas, um inteiro ASCII correspondente a uma tecla ou um código virtual. Se um caractere entre aspas for utilizado, ele será assumido como sendo um caractere ASCII. Se for um inteiro, deve-se informar ao compilador de recursos que ele é um

caractere ASCII especificando **Type** como ASCII. Caso seja uma tecla virtual, **Type** deverá ser declarado como VIRTKEY.

Se a tecla for um caractere maiúsculo entre aspas, então o item de menu correspondente será selecionado se a tecla for pressionada em conjunto com a tecla **SHIFT**. Se a tecla especificada for um caractere minúsculo e **ALT** for especificada em **Option**, então o item do menu correspondente só será selecionado com o pressionamento da tecla **ALT** em conjunto com o caractere especificado. Se o caractere for maiúsculo, então é necessário pressionar **SHIFT**, **ALT** e a tecla do caractere. Para utilizar a tecla **Ctrl**, utiliza-se o acento circunflexo '^' antes da letra que especifica a tecla aceleradora, como a seguir:

```
"^(letra)", IDM_...
```

ou a declaração de **Option** como **CONTROL** da forma:

```
"(letra)", IDM_..., CONTROL
```

Uma tecla virtual é um código independente do sistema para uma variedade de teclas. Várias teclas virtuais são predefinidas em **WINDOWS.H**, como as teclas de função **F1** a **F12**, as teclas de navegação, entre outras. Toda tecla virtual é definida como **VK\_...**, como por exemplo a utilizada no programa **SEDCURSO.RC**:

```
VK_F1, IDM_SOBRE, VIRTKEY
```

que especifica a utilização da tecla de função **F1** para apresentar a mensagem **SOBRE** numa caixa de diálogo.

Sempre que se utiliza uma tecla virtual, define-se **Type** como **VIRTKEY**. Também se pode utilizar **ALT**, **SHIFT** ou **CONTROL** para conseguir a combinação de teclas desejadas.

#### 4. INCLUINDO UM ÍCONE PERSONALIZADO

Geralmente os programas **WINDOWS** têm um ícone associado. Um ícone é uma figura em bitmap que aparece na barra de tarefas do **WINDOWS** e na extremidade esquerda da barra de título dos programas. Até o presente momento, o ícone utilizado no programa **SEDCURSO** é o ícone predefinido do **WINDOWS** definido por **IDI\_APPLICATION** no registro da janela:

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

Como normalmente é de interesse ter um ícone personalizado no programa, pode-se construir este através do Borland Resource WorkShop um ícone, o qual pode ser definido como um arquivo do tipo **.ICO** ou binário. Em ambos os casos, este é incluído no arquivo de recursos **.RC**, porém de formas diferentes.

Quando se deseja incluir um ícone personalizado no programa, as seguintes linhas de código são incluídas:

```
classejan.hIcon = LoadIcon (hCopia, szNomeAplic);
```

no registro da janela do código **SEDCURSO.C**;

```
sedcurso ICON "sedcurso.ico"
```

no código **SEDCURSO.RC**. Observe que a linha incluída no código **.C** é geral para qualquer programa, a qual carrega o ícone declarado no arquivo **.RC** com o formato da linha incluída no arquivo **.RC**. A declaração desta linha é a seguinte: nome do aplicativo (da mesma forma que se encontra declarada em **szNomeAplic**); macro **ICON** e nome do arquivo do ícone entre aspas. Para o caso do programa aqui estudado, o ícone declarado é visto na Figura 4.



Figura 4

Este ícone é criado, incluindo-se as linhas de código dos respectivos arquivos acima citados.

#### 5. DESENHANDO NA JANELA

O **WINDOWS** apresenta uma ótima interface gráfica, onde numa janela podem-se apresentar texto, gráficos ou figuras. Todos os recursos gráficos podem ser coloridos, como texto, fundo e linhas.

Devido à necessidade e à importância de se definir um ambiente com uma boa aparência, muitos recursos podem ser utilizados para implementar um aplicativo. Dentre os recursos gráficos da interface do WINDOWS (ou GDI – Graphic Device Interface), encontram-se as definições de canetas, utilizadas para desenhar linhas, sejam elas cheias, tracejadas, pontilhadas, entre outras; formas pré-definidas como retângulos e círculos; desenho de imagens definidas pelo usuário sobre a janela, como bitmaps; definições das cores utilizadas para preenchimento de áreas especificadas utilizando pincéis e outros.

Em sua maioria, os aplicativos WINDOWS se apresentam com possibilidades de modificação no tamanho da janela. Esta situação sempre gera uma chamada à mensagem WM\_PAINT, que repinta a janela. Porém, o conteúdo da janela deve ser guardado de alguma forma para ser reapresentado quando estas situações ocorrem. Se o conteúdo da janela não for salvo de alguma forma, a mensagem WM\_PAINT repinta a janela limpa com a cor de fundo definida na classe de janela. Uma das formas de solucionar este problema é utilizando-se dos recursos gráficos do WINDOWS que garantem que a repintura da janela não elimine qualquer imagem que antes estava apresentada. Várias formas são aplicadas à repintura da janela. Aqui, esta será feita através de uma função específica de desenho.

A seguir é apresentado o código do programa SEDCURSO.C incrementado das linhas de código para a criação de desenhos na janela, bem como as modificações dos arquivos: SEDCURSO.H e SEDCURSO.RC. As linhas modificadas ou introduzidas são apresentadas em texto não itálico com um caractere \* ao lado direito.

```
-----
                                SEDCURSO.C
-----

#include <windows.h>
#include <commctrl.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include "sedcurso.h"

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
void InitToolBar (void);
void display(HDC);

TBBUTTON    tbButtons [NUMBUTTONS];
HWND        tbwnd;
int         ToolBarActive;
UINT        wSelection = IDM_E1;
HMENU       hMenu;
HBRUSH      Brush[6];
HPEN        pen[3];
COLORREF    esquema[6];
int nlin = 0, xPontolns = 0, yPontolns = 0, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3 = 250, x4 = 100, x5 =
    25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE, mov = FALSE,
    movest = TRUE, brspeca = FALSE, npeca = FALSE, modif = FALSE;

char buff[300], lembrete[] = "Lembrete!";

char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND        hjan;
    MSG         msg;
    WNDCLASS    classejan;
    HACCEL      hAccel;

    classejan.style = CS_HREDRAW | CS_VREDRAW;
    classejan.lpfnWndProc = ProcJan;
    classejan.cbClsExtra = 0;
    classejan.cbWndExtra = 0;
    classejan.hInstance = hCopia;

```

```

classejan.hIcon          =LoadIcon (hCopia, szNomeAplic);
classejan.hCursor       =LoadCursor (NULL, IDC_ARROW);
classejan.hbrBackground =GetObject (WHITE_BRUSH);
classejan.lpszMenuName  =szNomeAplic;
classejan.lpszClassName =szNomeAplic;

if (!RegisterClass (&classejan)) return 0;

hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
                    NULL, hCopia, NULL);

hAccel = LoadAccelerators (hCopia, szNomeAplic);
InitToolBar ();
InitCommonControls();

tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
                        IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
                        0, 16, 16, sizeof(TBBUTTON));

ShowWindow (hjan, iCmdMostrar);
UpdateWindow (hjan);

while (GetMessage (&msg, NULL, 0, 0)) {
    if (!TranslateAccelerator(hjan, hAccel, &msg)){
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}

return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    int conta;
    char buff[20];
    LPTOOLTIPTTEXT TTtext;

    switch (iMsg) {
        case WM_CREATE:
            hMenu = GetMenu (hjan);
            Brush[0] = CreateSolidBrush (RGB(255,255,255));
            Brush[1] = CreateSolidBrush (RGB(255,0,255));
            Brush[2] = CreateSolidBrush (RGB(0,255,0));
            Brush[3] = CreateSolidBrush (RGB(123,234,0));
            Brush[4] = CreateSolidBrush (RGB(255,255,255));
            Brush[5] = CreateSolidBrush (RGB(255,0,0));
            pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));
            pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));
            pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));
            esquema[0] = RGB(255,255,255);
            esquema[1] = RGB(255,0,255);
            esquema[2] = RGB(0,255,0);
            esquema[3] = RGB(73,189,102);
            esquema[4] = RGB(165,19,25);
            esquema[5] = RGB(12,138,2);
            hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");
            return 0;

        case WM_NOTIFY:
            TTtext = (LPTOOLTIPTTEXT) lParam;
            if (TTtext->hdr.code == TTN_NEEDTEXT)
                switch (TTtext->hdr.idFrom){
                    case IDM_GE: TTtext->lpszText = "Girar Esteira";
                                break;
                    case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
                                break;
                    case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
                                break;
                    case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
                                break;
                    case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
                                break;
                }
    }
}

```

```

    }
    return 0;
case WM_COMMAND:
    switch (LOWORD(wParam)) {
        case IDM_ABRIR:
            MessageBox(hjan, "Não pode abrir o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_SALVAR:
            MessageBox(hjan, "Não pode salvar o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_IMPRIMIR:
            MessageBox(hjan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_GE:
            if(xest < 97)
                ge = TRUE;
            break;
        case IDM_PP:
            if (!brcpeca && (xest > 23)){
                pp = TRUE;
                fr = 0;
                pc = FALSE;
            }
            break;
        case IDM_GB:
            if (brcpeca)
                gb = TRUE;
            break;
        case IDM_SB:
            if (brcpeca && !gc && !gb && !pp){
                numpecas++;
                modif = TRUE;
                brcpeca = FALSE;
                fr = 0;
                MessageBeep(0xFFFFFFFF);
                InvalidateRect(hjan, NULL, TRUE);
            }
            break;
        case IDM_PT:
            sprintf(buf, "Número de Peças Trabalhadas: %d", numpecas);
            MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_E1:
        case IDM_E2:
        case IDM_E3:
            hMenu = GetMenu(hjan);
            CheckMenuItem(hMenu, wSelection, MF_UNCHECKED);
            wSelection = LOWORD(wParam);
            CheckMenuItem(hMenu, wSelection, MF_CHECKED);
            DeleteObject((HGDIOBJ) SetClassLong(hjan, GCL_HBRBACKGROUND, (LONG)
                Brush[wSelection/10 - 9]));
            InvalidateRect(hjan, NULL, TRUE);
            sprintf(buf, "Esquema de Cores %d", wSelection/10 - 8);
            MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_SAIR:
            if (IDOK == MessageBox(hjan, "Você deseja realmente terminar o programa?",
                szNomeAplic, MB_OKCANCEL | MB_ICONEXCLAMATION))
                SendMessage(hjan, WM_CLOSE, 0, 0L);
            break;
        case IDM_MOSTRARB:
            ToolBarActive = 1;
            ShowWindow(tbwnd, SW_RESTORE);
            InvalidateRect(hjan, NULL, 1);
            break;
        case IDM_ESCB:
            ToolBarActive = 0;
            ShowWindow(tbwnd, SW_HIDE);
            InvalidateRect(hjan, NULL, 1);
    }
}

```

```

        break;
    case IDM_SOBRE:
        MessageBox(hjan, "Programa de Simulação de Sistema\npara Aprendizagem de
        Programação\nem C para WINDOWS\n\nEduard Montgomery Meira Costa,
        DSc\n\n\t\t\t\t\t2002", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        return 0;
    }
    return 0;

case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);
    SelectObject(hdc, hFonte);
    SelectObject(hdc, Brush[wSelection/10 - 6]);
    SelectObject(hdc, pen[wSelection/10 - 9]);
    SetBkColor(hdc, esquema[wSelection/10 - 9]);
    SetTextColor (hdc, esquema[wSelection/10 - 6]);
    display (hdc);
    DeleteObject(pen[wSelection/10 - 9]);
    DeleteObject(Brush[wSelection/10 - 6]);
    DeleteObject((HGDIOBJ) hFonte);
    EndPaint (hjan, &ps);
    return 0;

case WM_CLOSE:
    DestroyWindow (hjan);
    return 0;

case WM_DESTROY:
    PostQuitMessage (0);
    return 0;
}
return DefWindowProc (hjan, iMsg, wParam, lParam);
}

void InitToolbar () {
tbButtons[0].iBitmap = 0;
tbButtons[0].idCommand = IDM_GE;
tbButtons[0].fsState = TBSTATE_ENABLED;
tbButtons[0].fsStyle = TBSTYLE_BUTTON;
tbButtons[0].dwData = 0;
tbButtons[0].iString = 0;

tbButtons[1].iBitmap = 0;
tbButtons[1].idCommand = 0;
tbButtons[1].fsState = TBSTATE_ENABLED;
tbButtons[1].fsStyle = TBSTYLE_SEP;
tbButtons[1].dwData = 0;
tbButtons[1].iString = 0;

tbButtons[2].iBitmap = 1;
tbButtons[2].idCommand = IDM_PP;
tbButtons[2].fsState = TBSTATE_ENABLED;
tbButtons[2].fsStyle = TBSTYLE_BUTTON;
tbButtons[2].dwData = 0;
tbButtons[2].iString = 0;

tbButtons[3].iBitmap = 2;
tbButtons[3].idCommand = IDM_GB;
tbButtons[3].fsState = TBSTATE_ENABLED;
tbButtons[3].fsStyle = TBSTYLE_BUTTON;
tbButtons[3].dwData = 0;
tbButtons[3].iString = 0;

tbButtons[4].iBitmap = 3;
tbButtons[4].idCommand = IDM_SB;
tbButtons[4].fsState = TBSTATE_ENABLED;
tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0;
tbButtons[4].iString = 0;

tbButtons[5].iBitmap = 0;
tbButtons[5].idCommand = 0;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0;
tbButtons[5].iString = 0;
}

```



```

tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0;
tbButtons[6].iString = 0;
}

void display(HDC hdc){
int i, j, k, cont;
char buffer[100], b[2];

//desenha a esteira
j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);
if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);
if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

//desenha o braço robótico
Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}
for(i = 0; i < 2; i++){
    MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
    MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
}
for(i = 0; i < 6; i++){
    MoveToEx (hdc, x1 + 20 - i, y1, NULL);
    LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
    MoveToEx (hdc, x1 - 20 + i, y1, NULL);
    LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
}
if(brcpeca)
    Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);
Ellipse (hdc, 225, 375, 275, 425);
Ellipse (hdc, 228, 378, 272, 422);
MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);

```

```

Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
MoveToEx (hdc, x2 - 25*cos(fi2), y2 - 25*sin(fi2), NULL);
LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

//desenha buffer de saída
for(cont = 0; cont < 3; cont++)
    Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);
sprintf(buffer, "Número de Peças: %d", numpeças);
TextOut(hdc, 370, 410, buffer, strlen(buffer));

//Apresenta os esquemas
TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
TextOut(hdc, 600, 68, "1", 1);
TextOut(hdc, 600, 86, "2", 1);
TextOut(hdc, 600, 104, "3", 1);

//Apresenta o texto do lembrete
for(i = 0; i < 5; i++){
    if(i%2 == 0){
        MoveToEx(hdc, i, 20 + i, NULL);
        LineTo(hdc, i, 165 - i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    MoveToEx(hdc, i, 30 + i, NULL);
    LineTo(hdc, 370 - i, 30 + i);
    LineTo(hdc, 370 - i, 165 - i);
}
i = 0;
k = 0;
TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
for(cont = 0; cont < strlen(buff); cont++){
    b[0] = buff[cont];
    b[1] = '\0';
    TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
    i++;
    if(i == 50){
        i = 0;
        k++;
    }
}
MoveToEx (hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
MoveToEx (hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);
}

```

---

## SEDCURSO.H

---

```

#define IDM_ABRIR          0
#define IDM_SALVAR         10
#define IDM_IMPRIMIR       20
#define IDM_SAIR           30
#define IDM_GE              40
#define IDM_PP              50
#define IDM_GB              60
#define IDM_SB              70
#define IDM_PT              80
#define IDM_E1              90
#define IDM_E2             100
#define IDM_E3             110
#define IDM_MOSTRARB        120
#define IDM_ESCB            130
#define IDM_SOBRE           140

#define IDTB_BMP            300
#define IDM_TOOLBAR         200
#define NUMBUTTONS          7

#define pi 3.1415

```

```
#include "sedcurso.h"
IDTB_BMP BITMAP "toolbarsedcurso.bmp"

sedcurso MENU {
  POPUP "&Arquivo" {
    MENUITEM "&Abrir\t^A", IDM_ABRIR
    MENUITEM "&Salvar\t^S", IDM_SALVAR
    MENUITEM "&Imprimir\t^I", IDM_IMPRIMIR
    MENUITEM SEPARATOR
    MENUITEM "Sai\r\t^R", IDM_SAIR
  }
  POPUP "&Sistema" {
    MENUITEM "&Girar Esteira\t^G", IDM_GE
    POPUP "&Braço Robótico" {
      MENUITEM "&Pegar Peça\t^P", IDM_PP
      MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
      MENUITEM "S&oltar Peça\t^O", IDM_SB
    }
    MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
  }
  POPUP "&Cores" {
    MENUITEM "Esquema &1\t^H", IDM_E1
    MENUITEM "Esquema &2\t^J", IDM_E2
    MENUITEM "Esquema &3\t^K", IDM_E3
  }
  POPUP "&Botões" {
    MENUITEM "&Mostrar\t^M", IDM_MOSTRARB
    MENUITEM SEPARATOR
    MENUITEM "Es&conder\t^C", IDM_ESCB
  }
  POPUP "&Ajuda" {
    MENUITEM "So&bre\t^F1", IDM_SOBRE
  }
}

sedcurso ACCELERATORS {
  "^A",    IDM_ABRIR
  "^S",    IDM_SALVAR
  "I",     IDM_IMPRIMIR, CONTROL
  "^R",    IDM_SAIR
  "^G",    IDM_GE
  "^P",    IDM_PP
  "^B",    IDM_GB
  "^O",    IDM_SB
  "^E",    IDM_PT
  "H",     IDM_E1, CONTROL
  "^J",    IDM_E2
  "^K",    IDM_E3
  "^M",    IDM_MOSTRARB
  "^C",    IDM_ESCB
  VK_F1,   IDM_SOBRE, VIRTKEY
}

sedcurso ICON "sedcurso.ico"
```

---

A inclusão/modificação do código no arquivo SEDCURSO.C e os arquivos SEDCURSO.H e SEDCURSO.RC geram a janela vista na Figura 5, na qual é desenhado todo o sistema (braço robótico, esteira e buffer de saída) em suas posições iniciais, quadro de lembrete vazio e esquemas de cores (texto para seleção do esquema de cor a ser utilizado) iniciado pelo esquema 1.

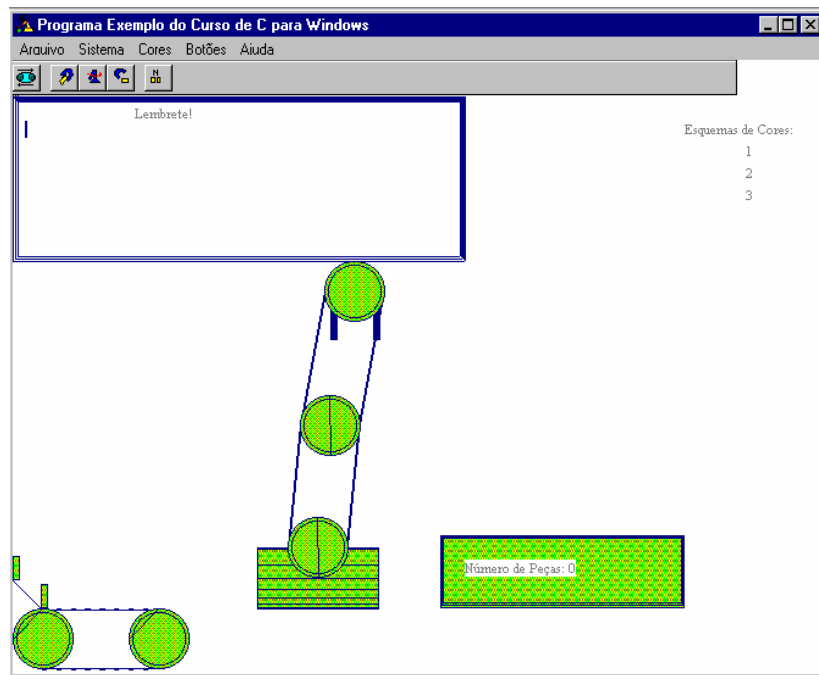


Figura 5

### As Mudanças/Inclusões no Programa SEDCURSO

As primeiras linhas incluídas no código do arquivo .C definem as inclusões dos arquivos de cabeçalho necessários pela utilização de algumas funções da linguagem C para trabalhar com funções matemáticas e de texto, como serão vistas adiante. São elas:

```
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
```

A seguir, é incluído o protótipo da função que apresenta todos os desenhos e textos na janela do programa:

```
void display(HDC);
```

As variáveis globais incluídas no programa são:

```
HBRUSH Brush[6];
HPEN pen[3];
COLORREF esquema[6];
int nlin = 0, xPontolns = 0, yPontolns = 0, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3=250,
    x4 = 100, x5 = 25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE,
    mov = FALSE, movest = TRUE, brspeca = FALSE, npeca = FALSE, modif = FALSE;

char buff[300], lembrete[] = "Lembrete!";
```

Essas variáveis são utilizadas da seguinte forma no programa:

- seis pincéis para pintar as figuras nos esquemas de cores;
- três canetas para traçar as linhas dos desenhos nos esquemas de cores;
- seis cores para os esquemas de cores para cor de fundo e cor da fonte;
- variáveis inteiras para: contagem de número de linhas no quadro de lembretes; posição  $x$  e  $y$  do cursor (ponto de inserção) de texto no quadro de lembrete; posição da mão do braço robótico; localização de peça na esteira; número de peças levadas ao buffer de saída; variáveis de posicionamento do braço robótico, da esteira e da peça na esteira ( $x_i$  e  $y_i$ ,  $i=1, 2, 3, 4, 5$ ) e posicionamento de nova peça na esteira;
- identificador para fonte;
- variáveis para posicionamento angular dos desenhos do braço robótico e da esteira;
- variáveis de teste para avaliação dos movimentos da esteira e do braço robótico e modificação do arquivo;

- variável para guardar o texto do lembrete e variável utilizada para apresentar o texto 'Lembrete!' no quadro de lembretes.

A função WinMain, para este caso, não contém nenhuma inclusão feita, como pode observar o leitor. Entretanto, o procedimento de janela inicialmente inclui as variáveis locais, definidas por:

```
HDC hdc;  
PAINTSTRUCT ps;  
RECT rect;  
int conta;  
char buf[20];
```

Essas variáveis são utilizadas para: obter o identificador do dispositivo (qual o periférico em que será apresentado o texto e gráficos: monitor, impressora); estrutura de pintura da janela; estrutura dos pontos da janela que guarda os valores dos pontos iniciais (top e left) e finais (bottom e right); variável para ser utilizada como contador e variável para ser utilizada em mensagens.

Com as definições das variáveis locais, é incluído no procedimento de janela, outras mensagens a serem processadas pelo WINDOWS. São elas:

- WM\_CREATE: mensagem WINDOWS geralmente utilizada para inicializar variáveis dentro do procedimento de janela.
- WM\_PAINT: mensagem utilizada para o WINDOWS repintar o conteúdo da janela quando ela é redimensionada. Esta mensagem necessita processar sempre as funções BeginPaint e EndPaint. Essas funções têm dois parâmetros que são: 1) hjan, que identifica a janela a ser repintada, e 2) o endereço da estrutura PAINTSTRUCT, definido por &ps, que recebe as informações de pintura.

No programa SEDCURSO, a mensagem WM\_CREATE é utilizada para inicializar o identificador do menu; os pincéis com as cores respectivas; as canetas; as cores dos esquemas de cores de fundo e fonte e o identificador da fonte a ser utilizada.

```
case WM_CREATE:  
    hMenu = GetMenu (hjan);  
    Brush[0] = CreateSolidBrush (RGB(255,255,255));  
    Brush[1] = CreateSolidBrush (RGB(255,0,255));  
    Brush[2] = CreateSolidBrush (RGB(0,255,0));  
    Brush[3] = CreateSolidBrush (RGB(123,234,0));  
    Brush[4] = CreateSolidBrush (RGB(255,255,255));  
    Brush[5] = CreateSolidBrush (RGB(255,0,0));  
    pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));  
    pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));  
    pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));  
    esquema[0] = RGB(255,255,255);  
    esquema[1] = RGB(255,0,255);  
    esquema[2] = RGB(0,255,0);  
    esquema[3] = RGB(73,189,102);  
    esquema[4] = RGB(165,19,25);  
    esquema[5] = RGB(12,138,2);  
    hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");  
    return 0;
```

A função CreateSolidBrush cria um pincel com a cor definida pela função RGB(red, green, blue), em que os valores de red, green e blue variam de 0 a 255. A função CreatePen cria uma caneta com estilo definido no primeiro parâmetro, largura da linha e cor respectiva. Os estilos de linha que podem ser utilizados nos programas que utilizam gráficos podem ser: PS\_SOLID – caneta sólida; PS\_DASH – caneta tracejada; PS\_DOT – caneta pontilhada; PS\_DASHDOT – caneta com tracejado e pontilhado (um traço seguido de um ponto); PS\_DASHDOTDOT – caneta com tracejado e duplo pontilhado (um traço seguido de dois pontos). A função CreateFont obtém o identificador para a fonte a ser utilizada, a qual é definida em seus parâmetros. Os principais parâmetros desta função para criar fontes TrueType são:

- (1) tamanho ou altura da fonte;
- (2) largura da fonte – o valor zero pode ser utilizado para definir automaticamente a largura relativa à altura;
- (5) atributo de espessura da linha da fonte. Pode ter valores de 0 (padrão) a 1000. Seus valores definidos no WINDOWS.H são - FW\_DONTCARE 0; FW\_THIN 100; FW\_EXTRALIGHT 200; FW\_ULTRALIGHT 200; FW\_LIGHT 300 FW\_NORMAL

400; FW\_REGULAR 400; FW\_MEDIUM 500; FW\_SEMIBOLD 600; FW\_DEMIBOLD 600; FW\_BOLD 700; FW\_EXTRABOLD 800; FW\_ULTRABOLD 800; FW\_HEAVY 900 e FW\_BLACK 900;

- (6) atributo de formato itálico. Se TRUE é itálico, se FALSE é normal;
- (7) atributo de sublinhado. Igual ao atributo de formato itálico;
- (8) atributo de caractere riscado. Igual ao atributo de formato itálico e sublinhado;
- (13) nome da família da fonte como, por exemplo: Courier New, Arial, Times New Roman.

Os demais parâmetros definem características de ângulo de inclinação, identificador de conjunto de caracteres, precisão e qualidade de saída. Estes valores podem ser definidos como zero para trabalhar com fontes TrueType.

A mensagem WM\_COMMAND, já vista anteriormente, inclui agora algumas modificações/inclusões para o processamento de alguns itens do menu:

```
case IDM_GE:
    if(xest < 97)
        ge = TRUE;
    break;
```

- caso seja selecionado “girar esteira”, se a a variável que define a localização da peça nela é menor que 97 (posição onde o braço robótico pode pegar a peça), então torna-se a variável ge verdadeira (variável que determina que a esteira pode girar);

```
case IDM_PP:
    if (!brcpeca && (xest > 23)){
        pp = TRUE;
        fr = 0;
        pc = FALSE;
    }
    break;
```

- caso seja selecionado “pegar peça”, se o braço robótico estiver livre e a peça na esteira estiver se guiando à posição em que o braço pode pegá-la (ou estiver na posição exigida x=97), torna a variável pp verdadeira, informando que o braço robótico pode se guiar para a estira para pegar a peça; define que a mão do braço robótico deve se tornar aberta e torna a variável de teste de posição do braço robótico falsa. A variável pc é utilizada para poder gerar o movimento correto do braço de acordo com sua posição em relação ao seu centro.

```
case IDM_GB:
    if (brcpeca)
        gb = TRUE;
    break;
```

- caso seja selecionado “girar braço”, se o braço robótico estiver com peça gb torna-se verdadeira, informando que o braço robótico pode ser girado para levar a peça ao buffer de saída.

```
case IDM_SB:
    if (brcpeca && !gc && !gb && !pp){
        numpecas++;
        modif = TRUE;
        brcpeca = FALSE;
        fr = 0;
        MessageBeep(0xFFFFFFFF);
        InvalidateRect(hjan, NULL, TRUE);
    }
    break;
```

- caso seja selecionado “soltar peça”, se o braço robótico estiver com peça e estiver posicionado sobre o buffer de saída (a variável gc é utilizada para testar o posicionamento do braço robótico em relação ao seu centro – para o movimento contrário ao da variável pc; a variável gb testa se o braço robótico já está parado – posicionado sobre o buffer; e pp se o braço robótico não está indo pegar peça), o número de peças no buffer de saída é incrementado; o arquivo é considerado como modificado; o braço robótico passa a ter o valor de “sem peça”; a variável fr é tornada zero para a mão do braço robótico se tornar aberta; um bip é soado no auto-falante do computador através da função MessageBeep (esta função gera um som na saída do auto-falante do computador, ou nas caixas de som, dependendo do valor definido em seu parâmetro, o qual pode ser: 0xFFFFFFFF –

som padrão usando o auto-falante do computador; MB\_ICONASTERISK, MB\_ICONEXCLAMATION, MB\_ICONHAND, MB\_ICONQUESTION, MB\_OK – que são sons do sistema).

```
case IDM_PT:
    sprintf (buf, "Número de Peças Trabalhadas: %d", numpecas);
    MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
    break;
```

- caso seja selecionado “contar peças”, coloca na variável buf o texto ‘Número de peças trabalhadas:’ com o número de numpecas que contém a informação do número de peças levadas ao buffer de saída.

Quando é selecionado um dos itens de menu dos esquemas de cores,

```
case IDM_E1:
case IDM_E2:
case IDM_E3:
```

o item é selecionado (como já visto anteriormente,

```
hMenu = GetMenu(hjan);
CheckMenuItem (hMenu, wSelection, MF_UNCHECKED);
wSelection = LOWORD(wParam);
CheckMenuItem (hMenu, wSelection, MF_CHECKED);
```

o fundo da janela é pintado com o pincel especificado com o esquema:

```
DeleteObject ((HGDIOBJ) SetClassLong (hjan, GCL_HBRBACKGROUND, (LONG)
    Brush[wSelection/10 - 9]));
```

a janela é repintada

```
InvalidateRect(hjan, NULL, TRUE);
```

e uma mensagem é apresentada com o esquema de cor escolhido:

```
sprintf (buf, "Esquema de Cores %d", wSelection/10 - 8);
MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
break;
```

A função DeleteObject destrói um objeto selecionado (no caso, o último pincel definido para a pintura do fundo da janela – WHITE\_BRUSH inicialmente no registro de classe da janela, ou outro definido posteriormente com a seleção de um esquema). A função SetClassLong determina que a janela deve ter o item da classe da janela referente ao segundo parâmetro modificado, devendo utilizar o novo valor definido pelo seu identificador (LONG). A declaração (LONG), antes do identificador determina a transformação de seu valor em um valor do tipo *longo*, que é o tipo utilizado pela função SetClassLong. A função SetClassLong pode ser utilizada para modificar outros valores de itens da classe da janela, utilizando para isto macros como:

```
GCL_HCURSOR para modificar cursores do mouse;
GCL_HICON para modificar ícones;
GCL_MENUNAME para modificar o menu da janela;
GCL_STYLE para modificar o estilo da classe da janela e
GCL_WNDPROC para modificar o endereço do procedimento da janela.
```

A mensagem WM\_PAINT é a mensagem enviada sempre que a função InvalidateRect é processada ou quando a janela é redimensionada (ou é reapresentada, caso esteja por trás de uma outra janela). Ela se processa da seguinte forma:

```
case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);
```

obtem o identificador do contexto do dispositivo (monitor);

```
SelectObject(hdc, hFonte);
```

seleciona a fonte a ser utilizada para a impressão de texto;

```
SelectObject(hdc, Brush[wSelection/10 - 6]);
```

seleciona o pincel para pintar o fundo das figuras;

```
SelectObject(hdc, pen[wSelection/10 - 9]);
```

seleciona a caneta para desenhar linhas;

```
SetBkColor(hdc, esquema[wSelection/10 - 9]);
```

define a cor de fundo do texto com a cor do esquema selecionado no menu;

```
SetTextColor (hdc, esquema[wSelection/10 - 6]);
```

define a cor da fonte;

```
display (hdc);
```

chama a função de apresentação dos desenhos e textos na janela, a qual após ser processada

```

DeleteObject(pen[wSelection/10 - 9]);
DeleteObject(Brush[wSelection/10 - 6]);
DeleteObject((HGDIOBJ) hFonte);

```

elimina todas os itens selecionados (obrigatoriamente isto sempre tem de ser feito para não estourar a pilha do computador) e

```

EndPaint (hjan, &ps);
return 0;

```

finaliza a chamada da mensagem.

A função display introduzida no programa se processa da seguinte forma: define as variáveis de contagem e de apresentação de texto:

```

int i, j, k, cont;
char buffer[100], b[2];

```

desenha a esteira gerando aleatoriamente um valor para simular seu movimento através do laço for:

```

j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);

```

A função MoveToEx posiciona o início de uma linha a ser traçada. Seus parâmetros são: identificador do contexto do dispositivo, posição  $x$ , posição  $y$  e o endereço da última posição (pode ser declarada como NULL e sempre redirecionar o início da posição da linha). A função LineTo desenha uma linha utilizando como parâmetros o identificador do contexto do dispositivo, e as posições  $x$  e  $y$  finais. A função Ellipse desenha uma elipse dentro de um quadrado definido pelos quatro parâmetros finais:  $x$  inicial,  $y$  inicial,  $x$  final e  $y$  final. As linhas de código que contém equações matemáticas em seno e cosseno (com a variável  $fi4$ ) desenharam uma linha que gira na elipse para simular o giro da esteira. Por fim, se a esteira está girando ou com uma peça na extremidade direita,

```

if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);

```

desenha a peça no respectivo lugar e se uma nova peça pode ser introduzida na esteira

```

if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

```

desenha a peça sendo introduzida. A função Rectangle desenha um retângulo com os mesmos parâmetros da função Ellipse.

Para desenhar o braço robótico, a função display inicialmente desenha sua base que é um retângulo com os devidos “enfeites”:

```

Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}

```

desenha as linhas das partes do braço robótico em suas respectivas posições:

```

for(i = 0; i < 2; i++){

```



```

MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
}

```

desenha a mão do braço robótico (aberta ou fechada – ou fechando):

```

for(i = 0; i < 6; i++){
    MoveToEx (hdc, x1 + 20 - i, y1, NULL);
    LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
    MoveToEx (hdc, x1 - 20 + i, y1, NULL);
    LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
}

```

se o braço robótico estiver com peça, desenha a peça em sua mão:

```

if(brcpeca)
    Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);

```

e desenha as junções entre as partes do braço robótico com as respectivas linhas de simulação de movimento:

```

Ellipse (hdc, 225, 375, 275, 425);
Ellipse (hdc, 228, 378, 272, 422);
MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);
Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
MoveToEx (hdc, x2 - 25*cos (fi2), y2 - 25*sin(fi2), NULL);
LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

```

O buffer de saída é desenhado como sendo um único retângulo com alguns “enfeites”:

```

for(cont = 0; cont < 3; cont++)
    Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);

```

e com o texto “Número de peças: ...”, onde as reticências determina o número de peças levadas à ele.

```

sprintf(buffer, "Número de Peças: %d", numpecas);
TextOut(hdc, 370, 410, buffer, strlen(buffer));

```

A função TextOut apresenta texto na área do cliente (parte da janela utilizada pelo usuário), tendo como parâmetros: o identificador do contexto do dispositivo, posições  $x$  e  $y$  iniciais onde será apresentado o texto, um ponteiro para o texto a ser apresentado e o número de caracteres do texto.

Por fim, a função display apresenta localizado à direita da janela os esquemas de cores:

```

TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
TextOut(hdc, 600, 68, "1", 1);
TextOut(hdc, 600, 86, "2", 1);
TextOut(hdc, 600, 104, "3", 1);

```

e o quadro do lembrete

```

for(i = 0; i < 5; i++){
    if(i%2 == 0){
        MoveToEx(hdc, i, 20 + i, NULL);
        LineTo(hdc, i, 165 - i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    MoveToEx(hdc, i, 30 + i, NULL);
    LineTo(hdc, 370 - i, 30 + i);
    LineTo(hdc, 370 - i, 165 - i);
}

```

com seu respectivo texto, caractere a caractere:

```

i = 0;
k = 0;
TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
for(cont = 0; cont < strlen(buff); cont++){
    b[0] = buff[cont];
    b[1] = '\0';
    TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
    i++;
}

```

```

if(i == 50){
    i = 0;
    k++;
}
}

```

e o ponto de inserção de texto (cursor de texto) é desenhado:

```

MoveToEx(hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
MoveToEx(hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);

```

No arquivo SEDCURSO.H, apenas é introduzida a definição da constante pi, que é utilizada para cálculos nas posições do braço robótico e esteira em seus movimentos.

```
#define pi 3.1415
```

No arquivo SEDCURSO.RC não há modificações.

## 6. UTILIZANDO O MOUSE

O mouse é um dos acessórios mais usados para operar programas WINDOWS. Com ele pode-se desenhar, definir localizações de cursor para escrever texto, fechar o programa, minimizar ou maximizar e escolher itens de menu.

O WINDOWS contém vários cursores para o mouse definido no arquivo de cabeçalho WINDOWS.H como o cursor padrão (seta), o cursor de espera de processamento (ampulheta), cursor de texto (barra vertical), entre outros. Porém, em um programa WINDOWS, pode-se criar um cursor do mouse para cada função desenvolvida. Por exemplo, quadrado, triângulo, computador, disquete, etc.

O mouse apresenta várias mensagens WINDOWS para seu processamento, como: WM\_LBUTTONDOWN, WM\_LBUTTONUP, WM\_LBUTTONDOWNBLCLK, WM\_RBUTTONDOWN, WM\_RBUTTONUP, WM\_RBUTTONDOWNBLCLK e WM\_MOUSEMOVE. As três primeiras, referem-se ao pressionamento, liberação e duplo pressionamento do botão esquerdo, respectivamente. As três posteriores, ao pressionamento, liberação e duplo pressionamento do botão direito, respectivamente. A sétima mensagem, refere-se ao movimento do mouse sobre a área do cliente.

Nas seis primeiras mensagens, os valores de LOWORD(IParam) e HIWORD(IParam) contém os valores  $x$  e  $y$  do mouse na área do cliente, podendo ser utilizadas para processar funções no programa que necessitam da localização do mouse. Exemplo disto é dado por programas de desenho, onde é necessário saber em que posição da área do cliente o botão do mouse foi pressionado para marcar um ponto. Por outro lado, quando desenhando linhas, o pressionamento do botão do mouse define a posição inicial, e quando se libera o botão, tem-se a posição final da linha.!

A seguir é apresentado o código do programa SEDCURSO.C incrementado das linhas de código para a utilização do mouse, bem como as modificações dos arquivos: SEDCURSO.H e SEDCURSO.RC. As linhas modificadas ou introduzidas são apresentadas em texto não itálico com um caractere \* ao lado direito.

---

### SEDCURSO.C

---

```

#include <windows.h>
#include <commctrl.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include "sedcurso.h"

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
void InitToolBar (void);
void display(HDC);

TBBUTTON    tbButtons [NUMBUTTONS];
HWND        tbwnd;
int         ToolBarActive;
UINT        wSelection = IDM_E1;

```

```

HMENU          hMenu;
HBRUSH Brush[6];
HPEN pen[3];
COLORREF esquema[6];
int nlin = 0, xPontolns = 0, yPontolns = 0, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3 = 250, x4 = 100, x5 =
    25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE, mov = FALSE,
    movest = TRUE, brcpeca = FALSE, npeca = FALSE, modif = FALSE;

char buff[300], lembrete[] = "Lembrete!";

char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND          hjan;
    MSG           msg;
    WNDCLASS      classejan;
    HACCEL        hAccel;

    classejan.style          =CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    classejan.lpfnWndProc     =ProcJan;
    classejan.cbClsExtra      =0;
    classejan.cbWndExtra      =0;
    classejan.hInstance      =hCopia;
    classejan.hIcon           =LoadIcon (hCopia, szNomeAplic);
    classejan.hCursor         =LoadCursor (NULL, IDC_ARROW);
    classejan.hbrBackground   =GetStockObject (WHITE_BRUSH);
    classejan.lpszMenuName     =szNomeAplic;
    classejan.lpszClassName   =szNomeAplic;

    if (!RegisterClass (&classejan)) return 0;

    hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
        NULL, hCopia, NULL);

    hAccel = LoadAccelerators (hCopia, szNomeAplic);
    InitToolBar ();
    InitCommonControls();

    tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
        IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
        16, 16, sizeof(TBBUTTON));

    ShowWindow (hjan, iCmdMostrar);
    UpdateWindow (hjan);

    while (GetMessage (&msg, NULL, 0, 0)) {
        if (!TranslateAccelerator(hjan, hAccel, &msg)){
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }

    return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    int conta;
    int xmouse, ymouse;
    char buf[20];
    LPTOOLTIPTTEXT TTtext;

    switch (iMsg) {
        case WM_CREATE:
            hMenu = GetMenu (hjan);
            Brush[0] = CreateSolidBrush (RGB(255,255,255));
            Brush[1] = CreateSolidBrush (RGB(255,0,255));
            Brush[2] = CreateSolidBrush (RGB(0,255,0));
            Brush[3] = CreateSolidBrush (RGB(123,234,0));
            Brush[4] = CreateSolidBrush (RGB(255,255,255));
            Brush[5] = CreateSolidBrush (RGB(255,0,0));

```

```

pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));
pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));
pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));
esquema[0] = RGB(255,255,255);
esquema[1] = RGB(255,0,255);
esquema[2] = RGB(0,255,0);
esquema[3] = RGB(73,189,102);
esquema[4] = RGB(165,19,25);
esquema[5] = RGB(12,138,2);
hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");
return 0;

case WM_NOTIFY:
    TTtext = (LPTOOLTIPTTEXT) lParam;
    if (TTtext->hdr.code == TTN_NEEDTEXT)
        switch (TTtext->hdr.idFrom){
            case IDM_GE: TTtext->lpszText = "Girar Esteira";
                break;
            case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
                break;
            case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
                break;
            case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
                break;
            case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
                break;
        }
    return 0;

case WM_COMMAND:
    switch (LOWORD(wParam)) {
        case IDM_ABRIR:
            MessageBox(hJan, "Não pode abrir o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_SALVAR:
            MessageBox(hJan, "Não pode salvar o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_IMPRIMIR:
            MessageBox(hJan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_GE:
            if(xest < 97)
                ge = TRUE;
            break;
        case IDM_PP:
            if (!brcpeca && (xest > 23)){
                pp = TRUE;
                fr = 0;
                pc = FALSE;
            }
            break;
        case IDM_GB:
            if (brcpeca)
                gb = TRUE;
            break;
        case IDM_SB:
            if (brcpeca && !gc && !gb && !pp){
                numpecas++;
                modif = TRUE;
                brcpeca = FALSE;
                fr = 0;
                MessageBeep(0xFFFFFFFF);
                InvalidateRect(hJan, NULL, TRUE);
            }
            break;
        case IDM_PT:
            sprintf(buf, "Número de Peças Trabalhadas: %d", numpecas);
            MessageBox(hJan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_E1:
        case IDM_E2:
    }

```

```

case IDM_E3:
    hMenu = GetMenu(hjan);
    CheckMenuItem(hMenu, wSelection, MF_UNCHECKED);
    wSelection = LOWORD(wParam);
    CheckMenuItem(hMenu, wSelection, MF_CHECKED);
    DeleteObject ((HGDIOBJ) SetClassLong (hjan, GCL_HBRBACKGROUND, (LONG)
        Brush[wSelection/10 - 9]));
    InvalidateRect(hjan, NULL, TRUE);
    sprintf (buf, "Esquema de Cores %d", wSelection/10 - 8);
    MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
    break;
case IDM_SAIR:
    if (IDOK == MessageBox (hjan, "Você deseja realmente terminar o programa?",
        szNomeAplic, MB_OKCANCEL | MB_ICONEXCLAMATION))
        SendMessage (hjan, WM_CLOSE, 0, 0L);
    break;
case IDM_MOSTRAR:
    ToolBarActive = 1;
    ShowWindow(tbwnd, SW_RESTORE);
    InvalidateRect(hjan, NULL, 1);
    break;
case IDM_ESCB:
    ToolBarActive = 0;
    ShowWindow(tbwnd, SW_HIDE);
    InvalidateRect(hjan, NULL, 1);
    break;
case IDM_SOBRE:
    MessageBox(hjan, "\tPrograma de Simulação de Sistema\npara Aprendizado de
        Programação\nem C para WINDOWS!\n\nEduard Montgomery Meira Costa,
        DSc\n\n1992", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
    return 0;
}
return 0;

case WM_LBUTTONDOWN:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        SendMessage(hjan, WM_COMMAND, IDM_GE, 0L);
    else if((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PP, 0L);
    else if((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_GB, 0L);
    else if((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_SB, 0L);
    else if((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PT, 0L);
    return 0;

case WM_RBUTTONDOWN:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82))
        SendMessage(hjan, WM_COMMAND, IDM_E1, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100))
        SendMessage(hjan, WM_COMMAND, IDM_E2, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 104) && (ymouse <= 118))
        SendMessage(hjan, WM_COMMAND, IDM_E3, 0L);
    return 0;

case WM_MOUSEMOVE:
    xmouse = LOWORD(lParam);
    ymouse = HIWORD(lParam);
    if(((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82)) || ((xmouse <= 614)
        && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100)) || ((xmouse <= 614) && (xmouse >=
        600) && (ymouse >= 104) && (ymouse <= 118))){
        hCursor = LoadCursor(hCop, "sedcur2");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if((xmouse < 360) && (ymouse < 162) && (ymouse > 45)){

```

```

        hCursor = LoadCursor(hCop, "sedcur1");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if (((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        ||((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        ||((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450)) ||
        ((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450)) ||
        ||((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))) {
        hCursor = LoadCursor(hCop, "sedcur3");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else {
        hCursor = LoadCursor(NULL, IDC_ARROW);
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }

    return 0;

case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);
    SelectObject(hdc, hFonte);
    SelectObject(hdc, Brush[wSelection/10 - 6]);
    SelectObject(hdc, pen[wSelection/10 - 9]);
    SetBkColor(hdc, esquema[wSelection/10 - 9]);
    SetTextColor (hdc, esquema[wSelection/10 - 6]);
    display (hdc);
    DeleteObject(pen[wSelection/10 - 9]);
    DeleteObject(Brush[wSelection/10 - 6]);
    DeleteObject((HGDIOBJ) hFonte);
    EndPaint (hjan, &ps);
    return 0;

case WM_CLOSE:
    DestroyWindow (hjan);
    return 0;

case WM_DESTROY:
    PostQuitMessage (0);
    return 0;
}
return DefWindowProc (hjan, iMsg, wParam, lParam);
}

void InitToolbar () {
    tbButtons[0].iBitmap = 0;
    tbButtons[0].idCommand = IDM_GE;
    tbButtons[0].fsState = TBSTATE_ENABLED;
    tbButtons[0].fsStyle = TBSTYLE_BUTTON;
    tbButtons[0].dwData = 0;
    tbButtons[0].iString = 0;

    tbButtons[1].iBitmap = 0;
    tbButtons[1].idCommand = 0;
    tbButtons[1].fsState = TBSTATE_ENABLED;
    tbButtons[1].fsStyle = TBSTYLE_SEP;
    tbButtons[1].dwData = 0;
    tbButtons[1].iString = 0;

    tbButtons[2].iBitmap = 1;
    tbButtons[2].idCommand = IDM_PP;
    tbButtons[2].fsState = TBSTATE_ENABLED;
    tbButtons[2].fsStyle = TBSTYLE_BUTTON;
    tbButtons[2].dwData = 0;
    tbButtons[2].iString = 0;

    tbButtons[3].iBitmap = 2;
    tbButtons[3].idCommand = IDM_GB;
    tbButtons[3].fsState = TBSTATE_ENABLED;
    tbButtons[3].fsStyle = TBSTYLE_BUTTON;
    tbButtons[3].dwData = 0;
    tbButtons[3].iString = 0;

    tbButtons[4].iBitmap = 3;
    tbButtons[4].idCommand = IDM_SB;
    tbButtons[4].fsState = TBSTATE_ENABLED;

```

```

tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0l;
tbButtons[4].iString = 0;

tbButtons[5].iBitmap = 0;
tbButtons[5].idCommand = 0;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0l;
tbButtons[5].iString = 0;

tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0l;
tbButtons[6].iString = 0;
}

void display(HDC hdc){
int i, j, k, cont;
char buffer[100], b[2];

//desenha a esteira
j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);
if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);
if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

//desenha o braço robótico
Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}
for(i = 0; i < 2; i++){
    MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
    MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
}
for(i = 0; i < 6; i++){
    MoveToEx (hdc, x1 + 20 - i, y1, NULL);
    LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
}

```

```

        MoveToEx (hdc, x1 - 20 + i, y1, NULL);
        LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
    }
    if(brcpeca)
        Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);
    Ellipse (hdc, 225, 375, 275, 425);
    Ellipse (hdc, 228, 378, 272, 422);
    MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
    LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
    Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);
    Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
    MoveToEx (hdc, x2 - 25*cos (fi2), y2 - 25*sin(fi2), NULL);
    LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
    Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
    Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

//desenha buffer de saída
for(cont = 0; cont < 3; cont++){
    Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);
    sprintf(buffer, "Número de Peças: %d", numpeças);
    TextOut(hdc, 370, 410, buffer, strlen(buffer));
}

//Apresenta os esquemas
TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
TextOut(hdc, 600, 68, "1", 1);
TextOut(hdc, 600, 86, "2", 1);
TextOut(hdc, 600, 104, "3", 1);

//Apresenta o texto do lembrete
for(i = 0; i < 5; i++){
    if(i%2 == 0){
        MoveToEx(hdc, i, 20 + i, NULL);
        LineTo(hdc, i, 165 - i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    MoveToEx(hdc, i, 30 + i, NULL);
    LineTo(hdc, 370 - i, 30 + i);
    LineTo(hdc, 370 - i, 165 - i);
}
i = 0;
k = 0;
TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
for(cont = 0; cont < strlen(buff); cont++){
    b[0] = buff[cont];
    b[1] = '\0';
    TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
    i++;
    if(i == 50){
        i = 0;
        k++;
    }
}
MoveToEx (hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
MoveToEx (hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);
}

```

---

#### SEDCURSO.H

---

```

#define IDM_ABRIR          0
#define IDM_SALVAR         10
#define IDM_IMPRIMIR       20
#define IDM_SAIR           30
#define IDM_GE             40
#define IDM_PP             50
#define IDM_GB             60
#define IDM_SB             70
#define IDM_PT             80
#define IDM_E1             90
#define IDM_E2            100

```



```

#define IDM_E3                110
#define IDM_MOSTRARB         120
#define IDM_ESCB             130
#define IDM_SOBRE            140

#define IDTB_BMP              300

#define IDM_TOOLBAR           200

#define NUMBUTTONS            7

#define pi 3.1415

```

---

SEDCURSO.RC

---

```

#include "sedcurso.h"

IDTB_BMP BITMAP "toolbarsedcurso.bmp"

sedcurso MENU {
    POPUP "&Arquivo" {
        MENUITEM "&Abrir\t^A", IDM_ABRIR
        MENUITEM "&Salvar\t^S", IDM_SALVAR
        MENUITEM "&Imprimir\t^I", IDM_IMPRIMIR
        MENUITEM SEPARATOR
        MENUITEM "Sai\r\t^R", IDM_SAIR
    }
    POPUP "&Sistema" {
        MENUITEM "&Girar Esteira\t^G", IDM_GE
    }
    POPUP "&Braço Robótico" {
        MENUITEM "&Pegar Peça\t^P", IDM_PP
        MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
        MENUITEM "S&oltar Peça\t^O", IDM_SB
    }
    MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
}
POPUP "&Cores" {
    MENUITEM "Esquema &1\t^H", IDM_E1
    MENUITEM "Esquema &2\t^J", IDM_E2
    MENUITEM "Esquema &3\t^K", IDM_E3
}
POPUP "&Botões" {
    MENUITEM "&Mostrar\t^M", IDM_MOSTRARB
    MENUITEM SEPARATOR
    MENUITEM "Es&conder\t^C", IDM_ESCB
}
POPUP "&Ajuda" {
    MENUITEM "So&bre\t^F1", IDM_SOBRE
}
}

sedcurso ACCELERATORS {
    "^A", IDM_ABRIR
    "^S", IDM_SALVAR
    "I", IDM_IMPRIMIR, CONTROL
    "^R", IDM_SAIR
    "^G", IDM_GE
    "^P", IDM_PP
    "^B", IDM_GB
    "^O", IDM_SB
    "^E", IDM_PT
    "H", IDM_E1, CONTROL
    "^J", IDM_E2
    "^K", IDM_E3
    "^M", IDM_MOSTRARB
    "^C", IDM_ESCB
    VK_F1, IDM_SOBRE, VIRTKEY
}

sedcurso ICON "sedcurso.ico"

sedcur1 CURSOR "cur1.cur"
sedcur2 CURSOR "cur2.cur"
sedcur3 CURSOR "cur3.cur"

```

\*  
\*  
\*

A inclusão/modificação do código no arquivo SEDCURSO.C e os arquivos SEDCURSO.H e SEDCURSO.RC incluem três cursores de mouse que são vistos na Figura 6, os quais são mostrados sempre que o mouse passa por uma região específica da janela: cursor ‘lápiz’ para a região do quadro de lembretes; cursor ‘mão’ para as regiões da esteira, base do braço robótico e buffer de saída, que identificam posições onde pode ser pressionado um botão para acionar o movimento específico de cada um e, uma ‘lista de cores’ para os números dos esquemas de cores.



Figura 6

### As Mudanças/Inclusões no Programa SEDCURSO

A introdução do mouse, neste programa, tem sua primeira linha de código modificada no registro da classe da janela na função WinMain, definida por:

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

A inclusão de CS\_DBLCLKS nesta linha de código é obrigatória quando se necessita de utilizar o duplo pressionamento de botões do mouse. Se o programa não utiliza este recurso, esta macro pode ser eliminada. Para este programa específico, apenas esta linha é modificada em WinMain.

Para o procedimento de janela, inicialmente, é incluída a linha:

```
int xmouse, ymouse;
```

que define duas variáveis locais responsáveis por guardar os valores de  $x$  e  $y$  da posição do mouse na janela, sempre que o mouse se movimenta, ou que um botão qualquer seja pressionado ou liberado.

As mensagens WINDOWS incluídas para o tratamento do mouse no programa SEDCURSO são: WM\_LBUTTONDOWN (processa o pressionamento do botão esquerdo do mouse), WM\_RBUTTONDOWNBLCLK (processa o duplo pressionamento do botão direito do mouse) e WM\_MOUSEMOVE (processa o movimento do mouse).

Na primeira mensagem processada (case WM\_LBUTTONDOWN:), as variáveis xmouse e ymouse recebem os valores  $x$  e  $y$  da posição do mouse, através das linhas de código:

```
xmouse = LOWORD (IParam);
ymouse = HIWORD (IParam);
```

As funções LOWORD e HIWORD obtêm a metade inferior e a metade superior do valor de IParam (que é um inteiro de 32 bits).

Com estes valores obtidos, avalia-se qual a posição do mouse na janela e, envia-se uma mensagem à WM\_COMMAND por meio da função SendMessage para processar o respectivo item do menu referente à região onde o botão do mouse foi pressionado:

```
if((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
    SendMessage(hjan, WM_COMMAND, IDM_GE, 0L);
else if((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
    SendMessage(hjan, WM_COMMAND, IDM_PP, 0L);
else if((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450))
    SendMessage(hjan, WM_COMMAND, IDM_GB, 0L);
else if((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
    SendMessage(hjan, WM_COMMAND, IDM_SB, 0L);
else if((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))
    SendMessage(hjan, WM_COMMAND, IDM_PT, 0L);

return 0;
```

Quando o botão direiro do mouse é pressionado duplamente (case WM\_RBUTTONDOWNBLCLK:) o procedimento é similar a mensagem anteriormente descrita. Carrega os valores da posição do mouse na janela:

```
xmouse = LOWORD (IParam);
ymouse = HIWORD (IParam);
```

avalia se o mouse encontra-se sobre as regiões onde se encontram os números referentes aos esquemas de cores e envia uma mensagem à WM\_COMMAND por meio da função SendMessage para processar os itens respectivos:

```
if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82))
```

```

        SendMessage(hjan, WM_COMMAND, IDM_E1, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100))
        SendMessage(hjan, WM_COMMAND, IDM_E2, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 104) && (ymouse <= 118))
        SendMessage(hjan, WM_COMMAND, IDM_E3, 0L);
    return 0;

```

Por fim, para qualquer movimento que seja feito com o mouse na área do cliente, a mensagem WM\_MOUSEMOVE é processada, pegando os valores x e y de sua posição sobre a janela:

```

xmouse = LOWORD(IParam);
ymouse = HIWORD(IParam);

```

testa em qual região se encontra o mouse: se está sobre uma das posições dos números do esquema de cores

```

if(((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82)) || ((xmouse <= 614)
&& (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100)) || ((xmouse <= 614) && (xmouse >=
600) && (ymouse >= 104) && (ymouse <= 118))){
    hCursor = LoadCursor(hCop, "sedcur2");
    SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
}

```

se está sobre na região do quadro de lembrete

```

else if((xmouse < 360) && (xmouse < 162) && (ymouse > 45)){
    hCursor = LoadCursor(hCop, "sedcur1");
    SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
}

```

se está sobre uma das regiões de comando do braço robótico, esteira ou buffer de saída

```

else if (((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
||((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
||((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450)) ||
((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450)) ||
||((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))){
    hCursor = LoadCursor(hCop, "sedcur3");
    SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
}

```

ou nenhuma das anteriormente citadas

```

else {
    hCursor = LoadCursor(NULL, IDC_ARROW);
    SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
}

```

```

return 0;

```

Para cada uma dessas regiões testadas, o identificador de cursor é obtido através da função LoadCursor, que tem como parâmetros o identificador da cópia do programa e o nome identificador do cursor entre aspas, o qual está definido no arquivo de recursos (.RC). Quando deseja-se modificar o cursor para a seta usualmente utilizada, carrega-se o identificador do cursor com os parâmetros definidos por: NULL e IDC\_ARROW.

Com o identificador do cursor carregado, a função SetClassLong define o novo cursor a ser utilizado (GCL\_HCURSOR), como visto anteriormente.

No arquivo SEDCURSO.H não há nenhuma modificação neste caso. Entretanto, no arquivo SEDCURSO.RC são incluídas as linhas de código:

```

sedcur1 CURSOR "cur1.cur"
sedcur2 CURSOR "cur2.cur"
sedcur3 CURSOR "cur3.cur"

```

que se referem aos cursores introduzidos para serem utilizados no programa. Cada definição de cursor segue a forma dada: nome identificador seguido da macro CURSOR e o nome do arquivo do cursor entre aspas. Observe que as chamadas aos cursores pela função LoadCursor utiliza o nome identificador entre aspas, e não o nome do arquivo.

## 7. UTILIZANDO O TEMPORIZADOR: GERANDO ANIMAÇÃO

Em muitas aplicações são necessárias definições de intervalos de tempo para realizar tarefas no programa. Especialmente em programas de simulação/científicos é exigida a utilização de relógios, contadores de tempo, entre outras aplicações.

O WINDOWS apresenta um temporizador que trata a interrupção do hardware decrementando o valor do contador originalmente passado pela função `SetTimer`. Quando esse valor chega a zero, o WINDOWS coloca uma mensagem `WM_TIMER` na fila do aplicativo e redefine seu valor original.

O temporizador do WINDOWS tem a precisão de 54,925 ms. Isso implica em que:

1. Um aplicativo WINDOWS não pode receber mensagens `WM_TIMER` a uma taxa superior a 18,2 vezes por segundo quando estiver usando um único temporizador;
2. O intervalo de tempo que você especifica na função `SetTimer` é sempre arredondado para baixo para um número inteiro múltiplo de pulsos de relógio.

As mensagens `WM_TIMER` são semelhantes às mensagens `WM_PAINT`. Sua diferença está em que o WINDOWS não coloca múltiplas mensagens `WM_TIMER` na fila. Ele combina várias mensagens `WM_TIMER` em uma única. Dessa forma, o aplicativo não receberá várias delas de uma vez só, embora possa receber duas em uma sucessão rápida. O aplicativo não pode determinar o número de mensagens `WM_TIMER` faltantes que resultam desse processo. Assim, se a barra de título for utilizada para mover a janela, o relógio para de receber mensagens `WM_TIMER`. Quando o relógio recebe novamente o controle e avança para a hora correta, não é por ter recebido várias mensagens `WM_TIMER` de uma só vez, mas porque ele teve de determinar a hora exata naquele instante de tempo. As mensagens `WM_TIMER` apenas informam ao programa quando ele deve ser atualizado.

Para incluir um temporizador em um programa são necessárias algumas definições de variáveis que devem ser atualizadas constantemente no procedimento da janela para se ter o tempo correto.

A seguir é apresentado o código do programa `SEDCURSO` com as inclusões/modificações introduzidas para a utilização do temporizador. As linhas modificadas ou introduzidas são apresentadas em texto não itálico com um caractere `*` ao lado direito.

```
-----
                                SEDCURSO.C
-----

#include <windows.h>
#include <commctrl.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include "sedcurso.h"

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
void InitToolbar (void);
void display(HDC);

TBBUTTON      tbButtons [NUMBUTTONS];
HWND          tbwnd;
int           ToolBarActive;
UINT          wParam = IDM_E1;
HMENU         hMenu;
HBRUSH Brush[6];
HPEN pen[3];
COLORREF esquema[6];
int nlin = 0, xPontos = 0, yPontos = 0, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3 = 250, x4 = 100, x5 =
    25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE, mov = FALSE,
    movest = TRUE, brcpeca = FALSE, npeca = FALSE, modif = FALSE;

char buff[300], lembrete[] = "Lembrete!";
char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND      hjan;
    MSG       msg;
    WNDCLASS  classejan;
```

```

HACCEL          hAccel;

classejan.style      =CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
classejan.lpfnWndProc =ProcJan;
classejan.cbClsExtra  =0;
classejan.cbWndExtra  =0;
classejan.hInstance   =hCopia;
classejan.hIcon        =LoadIcon (hCopia, szNomeAplic);
classejan.hCursor      =LoadCursor (NULL, IDC_ARROW);
classejan.hbrBackground =GetStockObject (WHITE_BRUSH);
classejan.lpszMenuName  =szNomeAplic;
classejan.lpszClassName =szNomeAplic;

if (!RegisterClass (&classejan)) return 0;

hjan = CreateWindow (szNomeAplic, szTitulo, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP,
                    NULL, hCopia, NULL);

hAccel = LoadAccelerators (hCopia, szNomeAplic);

SetTimer (hjan, ID_TEMPORIZ, 50, NULL);

InitToolBar ();
InitCommonControls();

tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
                        IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
                        0, 16, 16, sizeof(TBBUTTON));

ShowWindow (hjan, iCmdMostrar);
UpdateWindow (hjan);

while (GetMessage (&msg, NULL, 0, 0)) {
    if (!TranslateAccelerator(hjan, hAccel, &msg)){
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}

return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    int conta;
    int xmouse, ymouse;
    char buff[20];
    LPTOOLTIPTEXT TTtext;

    switch (iMsg) {
        case WM_CREATE:
            hMenu = GetMenu (hjan);
            Brush[0] = CreateSolidBrush (RGB(255,255,255));
            Brush[1] = CreateSolidBrush (RGB(255,0,255));
            Brush[2] = CreateSolidBrush (RGB(0,255,0));
            Brush[3] = CreateSolidBrush (RGB(123,234,0));
            Brush[4] = CreateSolidBrush (RGB(255,255,255));
            Brush[5] = CreateSolidBrush (RGB(255,0,0));
            pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));
            pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));
            pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));
            esquema[0] = RGB(255,255,255);
            esquema[1] = RGB(255,0,255);
            esquema[2] = RGB(0,255,0);
            esquema[3] = RGB(73,189,102);
            esquema[4] = RGB(165,19,25);
            esquema[5] = RGB(12,138,2);
            hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");
            return 0;

        case WM_NOTIFY:
            TTtext = (LPTOOLTIPTEXT) lParam;
            if (TTtext->hdr.code == TTN_NEEDTEXT)
                switch (TTtext->hdr.idFrom){
                    case IDM_GE: TTtext->lpszText = "Girar Esteira";
                }
    }
}

```

```

        break;
    case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
        break;
    case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
        break;
    case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
        break;
    case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
        break;
    }
    return 0;
case WM_COMMAND:
    switch (LOWORD(wParam)) {
        case IDM_ABRIR:
            MessageBox(hjan, "Não pode abrir o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_SALVAR:
            MessageBox(hjan, "Não pode salvar o Arquivo", szNomeAplic, MB_OK);
            break;
        case IDM_IMPRIMIR:
            MessageBox(hjan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_GE:
            if(xest < 97)
                ge = TRUE;
            break;
        case IDM_PP:
            if (!brcpeca && (xest > 23)){
                pp = TRUE;
                fr = 0;
                pc = FALSE;
            }
            break;
        case IDM_GB:
            if (brcpeca)
                gb = TRUE;
            break;
        case IDM_SB:
            if (brcpeca && !gc && !gb && !pp){
                numpecas++;
                modif = TRUE;
                brcpeca = FALSE;
                fr = 0;
                MessageBeep(0xFFFFFFFF);
                InvalidateRect(hjan, NULL, TRUE);
            }
            break;
        case IDM_PT:
            sprintf(buf, "Número de Peças Trabalhadas: %d", numpecas);
            MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_E1:
        case IDM_E2:
        case IDM_E3:
            hMenu = GetMenu(hjan);
            CheckMenuItem(hMenu, wSelection, MF_UNCHECKED);
            wSelection = LOWORD(wParam);
            CheckMenuItem(hMenu, wSelection, MF_CHECKED);
            DeleteObject((HGDIOBJ) SetClassLong(hjan, GCL_HBRBACKGROUND, (LONG)
                Brush[wSelection/10 - 9]));
            InvalidateRect(hjan, NULL, TRUE);
            sprintf(buf, "Esquema de Cores %d", wSelection/10 - 8);
            MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            break;
        case IDM_SAIR:
            if (IDOK == MessageBox(hjan, "Você deseja realmente terminar o programa?",
                szNomeAplic, MB_OKCANCEL | MB_ICONEXCLAMATION))
                SendMessage(hjan, WM_CLOSE, 0, 0L);
            break;
    }
}

```

```

        case IDM_MOSTRAR:
            ToolBarActive = 1;
            ShowWindow(tbwnd, SW_RESTORE);
            InvalidateRect(hjan, NULL, 1);
            break;
        case IDM_ESCB:
            ToolBarActive = 0;
            ShowWindow(tbwnd, SW_HIDE);
            InvalidateRect(hjan, NULL, 1);
            break;
        case IDM_SOBRE:
            MessageBox(hjan, "Programa de Simulação de Sistema para Aprendizagem de
            Programação em C para WINDOWS!\n\nEduard Montgomery Meira Costa,
            DSc\n\n1992", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
            return 0;
    }
    return 0;

case WM_LBUTTONDOWN:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        SendMessage(hjan, WM_COMMAND, IDM_GE, 0L);
    else if((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PP, 0L);
    else if((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_GB, 0L);
    else if((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_SB, 0L);
    else if((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PT, 0L);

    return 0;

case WM_RBUTTONDOWN:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82))
        SendMessage(hjan, WM_COMMAND, IDM_E1, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100))
        SendMessage(hjan, WM_COMMAND, IDM_E2, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 104) && (ymouse <= 118))
        SendMessage(hjan, WM_COMMAND, IDM_E3, 0L);

    return 0;

case WM_MOUSEMOVE:
    xmouse = LOWORD(lParam);
    ymouse = HIWORD(lParam);
    if(((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82)) || ((xmouse <= 614)
    && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100)) || ((xmouse <= 614) && (xmouse >=
    600) && (ymouse >= 104) && (ymouse <= 118))){
        hCursor = LoadCursor(hCop, "sedcur2");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if((xmouse < 360) && (ymouse < 162) && (ymouse > 45)){
        hCursor = LoadCursor(hCop, "sedcur1");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if (((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
    ||((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
    ||((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450)) ||
    ((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
    ||((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))){
        hCursor = LoadCursor(hCop, "sedcur3");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else {
        hCursor = LoadCursor(NULL, IDC_ARROW);
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }

    return 0;

```

```

case WM_TIMER:
    if (pp && !pc){
        mov = TRUE;
        if(x2 > 190){
            x2-=2;
            x4-=2;
        }
        if(y2 > 300){
            y2-=4;
            y4-=4;
        }
        if(x1 > 140){
            x1-=4;
            x4-=4;
        }
        if(y1 > 190){
            y1-=4;
            y4-=4;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 <= 250) && (y2 <= 300) && (x1 <= 250) && (y1 <= 190))
            pc = TRUE;
        InvalidateRect (hjan, NULL, TRUE);
    }
    else if (pp && pc){
        mov = TRUE;
        if(x2 > 180){
            x2-=2;
            x4-=2;
        }
        if(y2 < 300){
            y2+=2;
            y4+=2;
        }
        if(x1 > 100){
            x1-=4;
            x4-=4;
        }
        if(y1 < 400){
            y1+=2;
            y4+=2;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 <= 180) && (y2 >= 300) && (x1 <= 100) && (y1 >= 400)){
            pc = FALSE;
            pp = FALSE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    if (!pp && !pc && mov){
        fr+=2;
        if (fr > 10) {
            mov = FALSE;
            brcpeca = TRUE;
            movest = TRUE;
            xest = 23;
            npeca = TRUE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }

```



```

    }
    if (gb && !gc){
        if(x2 < 250){
            x2+=2;
            x4+=2;
        }
        if(y2 > 300){
            y2-=2;
            y4-=2;
        }
        if(x1 < 250){
            x1+=2;
            x4+=2;
        }
        if(y1 > 190){
            y1-=4;
            y4-=4;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 == 250) && (y2 == 300) && (x1 == 250) && (y1 <= 190))
            gc = TRUE;
        InvalidateRect (hjan, NULL, TRUE);
    }
else if (gb && gc){
    if(x2 <= 350){
        x2+=2;
        x4+=2;
    }
    if(y2 <= 300){
        y2+=2;
        y4+=2;
    }
    if(x1 <= 450){
        x1+=4;
        x4+=4;
    }
    if(y1 <= 340){
        y1+=2;
        y4+=2;
    }
    if ((x3 - x2) != 0)
        fi3 = atan((y3 - y2)/(x3 - x2));
    else fi3 = pi/2;
    if ((x2 - x1) != 0)
        fi2 = atan((y2 - y1)/(x2 - x1));
    else fi2 = pi/2;
    if ((x2 >= 350) && (y2 >= 300) && (x1 >= 450) && (y1 >= 340)){
        gc = FALSE;
        gb = FALSE;
    }
    InvalidateRect (hjan, NULL, TRUE);
}
if(brcpeca && (xnp == 0) && (xest > 23)) npeca = TRUE;
if(npeca && (xest > 23) &&(xnp < 23)) {
    xnp++;
    ynp++;
    InvalidateRect(hjan, NULL, TRUE);
}
else if (xnp == 23) npeca = FALSE;
if(xest == 23) {
    npeca = FALSE;
    movest = TRUE;
    xnp = 0;
    ynp = 407;
}

```

```

        if (ge && movest){
            fi4 = fi4+pi/10;
            xest+=2;
            if (xest == 97) {
                movest = FALSE;
                ge = FALSE;
                npeca = TRUE;
            }
            InvalidateRect (hjan, NULL, TRUE);
        }
        return 0;

    case WM_PAINT:
        hdc = BeginPaint (hjan, &ps);
        SelectObject(hdc, hFonte);
        SelectObject(hdc, Brush[wSelection/10 - 6]);
        SelectObject(hdc, pen[wSelection/10 - 9]);
        SetBkColor(hdc, esquema[wSelection/10 - 9]);
        SetTextColor (hdc, esquema[wSelection/10 - 6]);
        display (hdc);
        DeleteObject(pen[wSelection/10 - 9]);
        DeleteObject(Brush[wSelection/10 - 6]);
        DeleteObject((HGDIOBJ) hFonte);
        EndPaint (hjan, &ps);
        return 0;

    case WM_CLOSE:
        DestroyWindow (hjan);
        return 0;

    case WM_DESTROY:
        PostQuitMessage (0);
        return 0;
    }
    return DefWindowProc (hjan, iMsg, wParam, lParam);
}

void InitToolbar () {
    tbButtons[0].iBitmap = 0;
    tbButtons[0].idCommand = IDM_GE;
    tbButtons[0].fsState = TBSTATE_ENABLED;
    tbButtons[0].fsStyle = TBSTYLE_BUTTON;
    tbButtons[0].dwData = 0l;
    tbButtons[0].iString = 0;

    tbButtons[1].iBitmap = 0;
    tbButtons[1].idCommand = 0;
    tbButtons[1].fsState = TBSTATE_ENABLED;
    tbButtons[1].fsStyle = TBSTYLE_SEP;
    tbButtons[1].dwData = 0l;
    tbButtons[1].iString = 0;

    tbButtons[2].iBitmap = 1;
    tbButtons[2].idCommand = IDM_PP;
    tbButtons[2].fsState = TBSTATE_ENABLED;
    tbButtons[2].fsStyle = TBSTYLE_BUTTON;
    tbButtons[2].dwData = 0l;
    tbButtons[2].iString = 0;

    tbButtons[3].iBitmap = 2;
    tbButtons[3].idCommand = IDM_GB;
    tbButtons[3].fsState = TBSTATE_ENABLED;
    tbButtons[3].fsStyle = TBSTYLE_BUTTON;
    tbButtons[3].dwData = 0l;
    tbButtons[3].iString = 0;

    tbButtons[4].iBitmap = 3;
    tbButtons[4].idCommand = IDM_SB;
    tbButtons[4].fsState = TBSTATE_ENABLED;
    tbButtons[4].fsStyle = TBSTYLE_BUTTON;
    tbButtons[4].dwData = 0l;
    tbButtons[4].iString = 0;

    tbButtons[5].iBitmap = 0;
    tbButtons[5].idCommand = 0;

```

```

tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0;
tbButtons[5].iString = 0;
tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0;
tbButtons[6].iString = 0;
}

void display(HDC hdc){
int i, j, k, cont;
char buffer[100], b[2];

//desenha a esteira
j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);
if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);
if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

//desenha o braço robótico
Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}
for(i = 0; i < 2; i++){
    MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
    MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
}
for(i = 0; i < 6; i++){
    MoveToEx (hdc, x1 + 20 - i, y1, NULL);
    LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
    MoveToEx (hdc, x1 - 20 + i, y1, NULL);
    LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
}
if(brcpeca)
    Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);
Ellipse (hdc, 225, 375, 275, 425);

```

```

Ellipse (hdc, 228, 378, 272, 422);
MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);
Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
MoveToEx (hdc, x2 - 25*cos (fi2), y2 - 25*sin(fi2), NULL);
LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

//desenha buffer de saída
for(cont = 0; cont < 3; cont++){
    Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);
    sprintf(buffer, "Número de Peças: %d", numpeças);
    TextOut(hdc, 370, 410, buffer, strlen(buffer));
}

//Apresenta os esquemas
TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
TextOut(hdc, 600, 68, "1", 1);
TextOut(hdc, 600, 86, "2", 1);
TextOut(hdc, 600, 104, "3", 1);

//Apresenta o texto do lembrete
for(i = 0; i < 5; i++){
    if(i%2 == 0){
        MoveToEx(hdc, i, 20 + i, NULL);
        LineTo(hdc, i, 165 - i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    MoveToEx(hdc, i, 30 + i, NULL);
    LineTo(hdc, 370 - i, 30 + i);
    LineTo(hdc, 370 - i, 165 - i);
}
i = 0;
k = 0;
TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
for(cont = 0; cont < strlen(buff); cont++){
    b[0] = buff[cont];
    b[1] = '\0';
    TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
    i++;
    if(i == 50){
        i = 0;
        k++;
    }
}
MoveToEx (hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
MoveToEx (hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);
}

```

---

#### SEDCURSO.H

---

```

#define IDM_ABRIR          0
#define IDM_SALVAR         10
#define IDM_IMPRIMIR      20
#define IDM_SAIR           30
#define IDM_GE             40
#define IDM_PP             50
#define IDM_GB             60
#define IDM_SB             70
#define IDM_PT             80
#define IDM_E1             90
#define IDM_E2            100
#define IDM_E3            110
#define IDM_MOSTRAR        120
#define IDM_ESCB           130
#define IDM_SOBRE          140
#define IDTB_BMP           300

```

```
#define IDM_TOOLBAR      200
#define NUMBUTTONS      7
#define ID_TEMPORIZ      1
#define pi 3.1415
```

---

### SEDCURSO.RC

---

```
#include "sedcurso.h"
IDTB_BMP BITMAP "toolbarsedcurso.bmp"

sedcurso MENU {
  POPUP "&Arquivo" {
    MENUITEM "&Abrir\t^A", IDM_ABRIR
    MENUITEM "&Salvar\t^S", IDM_SALVAR
    MENUITEM "&Imprimir\t^I", IDM_IMPRIMIR
    MENUITEM SEPARATOR
    MENUITEM "Sai\r\t^R", IDM_SAIR
  }
  POPUP "&Sistema" {
    MENUITEM "&Girar Esteira\t^G", IDM_GE
    POPUP "&Braço Robótico" {
      MENUITEM "&Pegar Peça\t^P", IDM_PP
      MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
      MENUITEM "S&oltar Peça\t^O", IDM_SB
    }
    MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
  }
  POPUP "&Cores" {
    MENUITEM "Esquema &1\t^H", IDM_E1
    MENUITEM "Esquema &2\t^J", IDM_E2
    MENUITEM "Esquema &3\t^K", IDM_E3
  }
  POPUP "&Botões" {
    MENUITEM "&Mostrar\t^M", IDM_MOSTRARB
    MENUITEM SEPARATOR
    MENUITEM "Es&conder\t^C", IDM_ESCB
  }
  POPUP "&Ajuda" {
    MENUITEM "So&bre\t^F1", IDM_SOBRE
  }
}

sedcurso ACCELERATORS {
  "^A", IDM_ABRIR
  "^S", IDM_SALVAR
  "I", IDM_IMPRIMIR, CONTROL
  "^R", IDM_SAIR
  "^G", IDM_GE
  "^P", IDM_PP
  "^B", IDM_GB
  "^O", IDM_SB
  "^E", IDM_PT
  "H", IDM_E1, CONTROL
  "^J", IDM_E2
  "^K", IDM_E3
  "^M", IDM_MOSTRARB
  "^C", IDM_ESCB
  VK_F1, IDM_SOBRE, VIRTKEY
}

sedcurso ICON "sedcurso.ico"
sedcur1 CURSOR "cur1.cur"
sedcur2 CURSOR "cur2.cur"
sedcur3 CURSOR "cur3.cur"
```

---

A inclusão/modificação do código no arquivo SEDCURSO.C e os arquivos SEDCURSO.H e SEDCURSO.RC inclui um temporizador que garante que o WINDOWS processe periodicamente a

mensagem WM\_TIMER, de forma que as variáveis são atualizadas, atualizando assim as posições dos desenhos na janela. Isto gera a animação do sistema apresentado na janela.

### As Mudanças/Inclusões no Programa SEDCURSO

Para trabalhar com o temporizador neste programa, é necessário definir o intervalo de tempo para as chamadas à mensagem WM\_TIMER. Esta é feita através da inclusão no código de WinMain da linha:

```
SetTimer (hjan, ID_TEMPORIZ, 50, NULL);
```

A função SetTimer inicializa o temporizador. Seus parâmetros são: identificador da janela; identificador do temporizador utilizado (constante definida em SEDCURSO.H); valor do intervalo de tempo utilizado (neste caso é 50 mili-segundos) e ponteiro para uma função que será notificada quando o valor do tempo é finalizado (o valor NULL garante que o sistema põe a mensagem WM\_TIMER na fila de mensagens do programa).

Com esta inclusão, o procedimento de janela inclui a mensagem WM\_TIMER, a qual é chamada a cada 50 mili-segundos. WM\_TIMER testa todas as variáveis em cada 50 mili-segundos passados, atualizando-as em função dos valores específicos para cada situação do braço robótico, esteira e peça. Todos os ângulos referentes ao braço robótico são calculados dependendo das posições em que ele se encontra, bem como os ângulos da esteira de forma similar (se é solicitado o movimento de um deles). Também, a mão do braço robótico tem sua variável (fr) calculada de forma similar, entre outras. Isto pode ser verificado seguindo as linhas do código pelo leitor.

```
case WM_TIMER:
    if (pp && !pc){
        mov = TRUE;
        if(x2 > 190){
            x2-=2;
            x4-=2;
        }
        if(y2 > 300){
            y2-=4;
            y4-=4;
        }
        if(x1 > 140){
            x1-=4;
            x4-=4;
        }
        if(y1 > 190){
            y1-=4;
            y4-=4;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 <= 250) && (y2 <= 300) && (x1 <= 250) && (y1 <= 190))
            pc = TRUE;
        InvalidateRect (hjan, NULL, TRUE);
    }
    else if (pp && pc){
        mov = TRUE;
        if(x2 > 180){
            x2-=2;
            x4-=2;
        }
        if(y2 < 300){
            y2+=2;
            y4+=2;
        }
        if(x1 > 100){
            x1-=4;
            x4-=4;
        }
        if(y1 < 400){
```

```

        y1+=2;
        y4+=2;
    }
    if ((x3 - x2) != 0)
        fi3 = atan((y3 - y2)/(x3 - x2));
    else fi3 = pi/2;
    if ((x2 - x1) != 0)
        fi2 = atan((y2 - y1)/(x2 - x1));
    else fi2 = pi/2;
    if ((x2 <= 180) && (y2 >= 300) && (x1 <= 100) && (y1 >= 400)){
        pc = FALSE;
        pp = FALSE;
    }
    InvalidateRect (hjan, NULL, TRUE);
}
if (!pp && !pc && mov){
    fr+=2;
    if (fr > 10) {
        mov = FALSE;
        brcpeca = TRUE;
        movest = TRUE;
        xest = 23;
        npeca = TRUE;
    }
    InvalidateRect (hjan, NULL, TRUE);
}
if (gb && !gc){
    if(x2 < 250){
        x2+=2;
        x4+=2;
    }
    if(y2 > 300){
        y2-=2;
        y4-=2;
    }
    if(x1 < 250){
        x1+=2;
        x4+=2;
    }
    if(y1 > 190){
        y1-=4;
        y4-=4;
    }
    if ((x3 - x2) != 0)
        fi3 = atan((y3 - y2)/(x3 - x2));
    else fi3 = pi/2;
    if ((x2 - x1) != 0)
        fi2 = atan((y2 - y1)/(x2 - x1));
    else fi2 = pi/2;
    if ((x2 == 250) && (y2 == 300) && (x1 == 250) && (y1 <= 190))
        gc = TRUE;
    InvalidateRect (hjan, NULL, TRUE);
}
else if (gb && gc){
    if(x2 <= 350){
        x2+=2;
        x4+=2;
    }
    if(y2 <= 300){
        y2+=2;
        y4+=2;
    }
    if(x1 <= 450){
        x1+=4;
        x4+=4;
    }
    if(y1 <= 340){
        y1+=2;
        y4+=2;
    }
}

```

```

        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 >= 350) && (y2 >= 300) && (x1 >= 450) && (y1 >= 340)){
            gc = FALSE;
            gb = FALSE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    if(brcpeca && (xnp == 0) && (xest > 23)) npeca = TRUE;
    if(npeca && (xest > 23) &&(xnp < 23)) {
        xnp++;
        ynp++;
        InvalidateRect(hjan, NULL, TRUE);
    }
    else if (xnp == 23) npeca = FALSE;
    if(xest == 23) {
        npeca = FALSE;
        movest = TRUE;
        xnp = 0;
        ynp = 407;
    }
    if (ge && movest){
        fi4 = fi4+pi/10;
        xest+=2;
        if (xest == 97) {
            movest = FALSE;
            ge = FALSE;
            npeca = TRUE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    return 0;

```

No arquivo SEDCURSO.H é incluída a linha de código de definição da constante do identificador do temporizador, que é:

```
#define ID_TEMPORIZ 1
```

O arquivo SEDCURSO.RC não apresenta nenhuma modificação para este caso.

## 8. UTILIZANDO O TECLADO: ESCRIVENDO TEXTO

Quando lidando com edição de texto, torna-se necessário utilizar vários recursos dos já estudados anteriormente. Entre eles, especialmente estão: o teclado, as fontes e as justificações de linhas. Nos editores de texto, geralmente se encontra também, a utilização do mouse para seleção de texto, além de barras de rolagem e outros.

A inclusão de um cursor no meio do texto, também é de grande necessidade, para poder se fazer modificações ou inclusões no texto que está sendo escrito.

Há várias maneiras de construir um programa de edição de texto. Dentre elas, é preciso conhecer como se processa alguns recursos do WINDOWS.

Nessa Seção, é apresentada a inclusão do código das mensagens WM\_KEYDOWN e WM\_CHAR no programa SEDCURSO, utilizadas para gerar o texto do lembrete. A fonte utilizada para isto é a “Times New Roman”, já formalizada anteriormente. O cursor atende a comando de pressionamento de teclas como BACKSPACE, apagando os caracteres para trás do cursor; DELETE, apagando os caracteres para frente do cursor; navega na janela para frente, para trás, para cima e para baixo, entre outras funções.

Como sendo a única modificação a ser introduzida no programa, em relação ao último código apresentado (Seção 7), a seguir são apresentadas apenas essas mensagens, que são incluídas no procedimento de janela, como mostrado nas seções anteriores.

```
case WM_KEYDOWN:
```



```

switch (wParam){
    case VK_HOME:
        xPontolns = 0;
        break;
    case VK_END:
        if(yPontolns == nlin)
            xPontolns = strlen(buff)%50;
        else xPontolns = 50;
        break;
    case VK_LEFT:
        xPontolns = xPontolns--;
        if((xPontolns == -1) && (yPontolns > 0)){
            xPontolns = 50;
            yPontolns--;
        }
        else if((xPontolns == -1) && (yPontolns == 0)) xPontolns = 0;
        break;
    case VK_RIGHT:
        xPontolns = xPontolns++;
        if ((xPontolns == 51) && (yPontolns < nlin)){
            xPontolns = 0;
            yPontolns++;
        }
        else if((yPontolns == nlin) && (xPontolns > strlen(buff)%50))
            xPontolns = strlen(buff)%50;
        break;
    case VK_UP:
        if (yPontolns > 0) yPontolns--;
        break;
    case VK_DOWN:
        if (yPontolns < nlin) yPontolns++;
        if((xPontolns > strlen(buff)%50) && (yPontolns == nlin))
            xPontolns = strlen(buff)%50;
        break;
    case VK_DELETE:
        for (conta = yPontolns*50 + xPontolns; conta < strlen(buff); conta++){
            buff[conta] = buff[conta + 1];
            buff[strlen(buff)] = '\0';
        }
        break;
    case VK_BACK:
        if (xPontolns > 0){
            xPontolns--;
            SendMessage (hjan, WM_KEYDOWN, VK_DELETE, 1L);
        }
        break;
    case VK_TAB:
        do{
            SendMessage (hjan, WM_CHAR, ' ', 1L);
        }while (xPontolns % 4 != 0);
        break;
    case VK_RETURN:
        for(conta = yPontolns*50 + xPontolns; conta < (yPontolns+1)*50; conta++){
            buff[conta + (yPontolns+1)*50 - 1] = buff[conta];
            buff[conta] = ' ';
        }
        xPontolns = 0;
        yPontolns++;
        if (yPontolns == 7){
            yPontolns = 6;
            MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
                MB_ICONEXCLAMATION);
        }
        break;
    }
    InvalidateRect(hjan, NULL, TRUE);
    return 0;
case WM_CHAR:
    if (strlen(buff) > 299){
        MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
            MB_ICONEXCLAMATION);
    }
}

```

```

        break;
    }
    else if(((isalnum(wParam))||!isalnum(wParam)) && ((char)wParam != '\b') && ((char)wParam != '\t')){
        for (conta = strlen(buff); conta > (yPontolns * 50 + xPontolns); conta--){
            buff[conta] = buff[conta - 1];
            buff[yPontolns * 50 + xPontolns] = (char) wParam;
            buff[strlen(buff) + 1] = '\0';
            xPontolns++;
            if(xPontolns == 50){
                xPontolns = 0;
                yPontolns++;
                if (yPontolns == 7){
                    yPontolns = 6;
                    MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
                        MB_ICONEXCLAMATION);
                    InvalidateRect (hjan, NULL, TRUE);
                }
            }
        }
        nlin = strlen(buff)/50;
        InvalidateRect (hjan, NULL, FALSE);
        return 0;
    }

```

Os demais arquivos (SEDCURSO.H e SEDCURSO.RC) não apresentam modificações em seus códigos.

### **Mensagens WM\_CHAR e WM\_KEYDOWN: Incluídas no Código de SEDCURSO**

A mensagem WM\_KEYDOWN é uma mensagem processada quando uma tecla é pressionada. Para esta mensagem, há vários códigos específicos para as teclas virtuais e para as demais teclas que podem ser avaliadas para realizar alguma função num programa. A exemplo das teclas virtuais, tem-se: VK\_HOME – tecla HOME, VK\_END – tecla END, VK\_PRIOR – tecla Page Up, VK\_NEXT – tecla Page Down, VK\_LEFT – tecla de navegação (seta) à esquerda, VK\_RIGHT – tecla de navegação (seta) à direita, VK\_UP – tecla de navegação (seta) para cima, VK\_DOWN – tecla de navegação (seta) para baixo, VK\_DELETE – tecla Delete, VK\_BACK – tecla BackSpace, VK\_RETURN – tecla Enter, etc. Neste programa, cada tecla utilizada realiza uma função específica para o texto do quadro de lembretes:

VK\_HOME – quando a tecla HOME é pressionada, o cursor de texto (ponto de inserção) vai para o início da linha em que ele se encontra;

VK\_END – quando a tecla END é pressionada, leva o cursor de texto para o final da linha em que ele se encontra;

VK\_LEFT – quando a tecla de navegação à esquerda é pressionada, o cursor de texto é levado um caractere para esquerda;

VK\_RIGHT – quando a tecla de navegação à direita é pressionada, o cursor de texto é levado um caractere para direita;

VK\_UP – quando a tecla de navegação acima é pressionada, o cursor de texto sobe uma linha;

VK\_DOWN – quando a tecla de navegação abaixo é pressionada, o cursor de texto desce uma linha;

VK\_DELETE – quando a tecla DELETE é pressionada, o caractere que se encontra ao lado direito do cursor de texto é eliminado, carregando todo o texto a partir de sua posição até o final, um caractere para a esquerda;

VK\_BACK – quando a tecla BackSpace é pressionada, o caractere que se encontra ao lado esquerdo do cursor de texto é eliminado, carregando todo o texto a partir de sua posição até o final, um caractere para a esquerda;

VK\_TAB – quando a tecla TAB é pressionada, é inserido uma tabulação de quatro espaços no texto;

VK\_RETURN – quando a tecla **Enter** é pressionada, é inserido um número de espaço suficiente para preencher a linha até o final, e transferir o cursor de texto para o início da outra linha.

Ao final da mensagem WM\_KEYDOWN a janela é repintada com o novo texto e com o cursor em sua nova posição

```
InvalidateRect(hjan, NULL, TRUE);
```

A mensagem WM\_CHAR é similar à mensagem WM\_KEYDOWN. Entretanto, nesta mensagem utilizam-se os caracteres específicos, ao invés das macros que definem as teclas. Por exemplo, o processamento da tecla **Enter** é definido pelas linhas:

```
case '\n':  
case '\r':
```

ao invés de

```
case VK_RETURN,
```

ou '\t' ao invés de VK\_TAB. Dessa forma, esta mensagem é mais utilizada para processar pressionamento de teclas de caracteres para texto (porém, não é necessariamente!). No programa SEDCURSO, tem-se para esta mensagem:

- se o número de caracteres ultrapassa o limite de 300 caracteres no texto do lembrete, uma mensagem avisa que não mais é possível colocar texto no lembrete e pára:

```
if (strlen(buff) > 299){  
    MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |  
        MB_ICONEXCLAMATION);  
    break;  
}
```

- caso contrário, testa se a tecla pressionada é caractere de texto ou numérico e se o caractere não é um backspace ou tabulação. Se isto for satisfeito, introduz o caractere digitado na posição em que se encontra o cursor de texto (ponto de inserção), sempre testando para cada tecla pressionada, se o número de caracteres ultrapassa 300, ou se o texto chega ao final da linha para passar para a próxima, ou se o texto ultrapassa 6 linhas:

```
else if(!isalnum(wParam)||!isalnum(wParam)) && ((char)wParam != '\b') && ((char)wParam != '\t')){  
    for (conta = strlen(buff); conta > (yPontolns * 50 + xPontolns); conta--)  
        buff[conta] = buff[conta - 1];  
    buff[yPontolns * 50 + xPontolns] = (char) wParam;  
    buff[strlen(buff) + 1] = '\0';  
    xPontolns++;  
    if(xPontolns == 50){  
        xPontolns = 0;  
        yPontolns++;  
        if (yPontolns == 7){  
            yPontolns = 6;  
            MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |  
                MB_ICONEXCLAMATION);  
            InvalidateRect (hjan, NULL, TRUE);  
        }  
    }  
}
```

Por fim, recalcula o número de linhas de texto e imprime o novo caractere digitado.

```
nlin = strlen(buff)/50;  
InvalidateRect (hjan, NULL, FALSE);  
return 0;
```

Deve-se observar que este procedimento é uma forma de colocar texto na janela, através da utilização do teclado. Existem várias formas para realizar isto, os quais podem ser consultados na bibliografia citada ao final desta apostila.

## 9. GRAVANDO EM DISCO

Em todo programa é necessário guardar os dados referentes ao que foi trabalhado em um dado momento, para em qualquer outro instante de tempo, estes dados poderem ser carregados pelo programa. São dados: imagens de bitmaps, texto (caracteres), números (inteiros ou não), posições de janela, entre outros.

Para que um programa WINDOWS execute gravação e abertura de arquivos, podem ser utilizadas as caixas de diálogo comuns de gravação e abertura. A gravação de um arquivo exige que todos os dados sejam passados em uma sequência, tal que sua abertura se processe da mesma maneira.

Tanto a gravação quanto a abertura de arquivos podem ser feitas com funções simples da linguagem C, como `fread` e `fwrite`. Entretanto, devem estar localizados dentro de procedimentos específicos do WINDOWS.

Aqui, o procedimento de gravação é incluído no programa SEDCURSO que está apresentado a seguir, onde as linhas incluídas/modificadas encontram-se em não itálico com um \* ao lado direito.

```
-----
                                SEDCURSO.C
-----

#include <windows.h>
#include <commctrl.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include "sedcurso.h"

LRESULT CALLBACK ProcJan (HWND, UINT, WPARAM, LPARAM);
void InitToolbar (void);
void display(HDC);
void InicializaArq (HWND);
BOOL PerguntaSeSalva (HWND, char *);
void PoeTitulo (HWND, char *);

HINSTANCE hCop;
HBRUSH Brush[6];
HPEN pen[3];
HCURSOR hCursor;
COLORREF esquema[6];
TBBUTTON tbButtons [NUMBUTTONS];
HWND tbwnd;
int nlin = 0, xPontoIns = 0, yPontoIns = 0, ToolBarActive, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3 = 250,
x4 = 100, x5 = 25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
UINT wSelection = IDM_E1;
HMENU hMenu;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE, mov = FALSE,
movest = TRUE, brpeca = FALSE, npeca = FALSE, modif = FALSE;
OPENFILENAME fname;
FILE *fp;
char filename[64] = SEMTITULO;
char buff[300], lembrete[] = "Lembrete!";
char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND hjan;
    MSG msg;
    WNDCLASS classejan;
    HACCEL hAccel;

    classejan.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    classejan.lpfnWndProc = ProcJan;
    classejan.cbClsExtra = 0;
    classejan.cbWndExtra = 0;
    classejan.hInstance = hCopia;
    classejan.hIcon = LoadIcon (hCopia, szNomeAplic);
    classejan.hCursor = LoadCursor (NULL, IDC_ARROW);
    classejan.hbrBackground = GetStockObject (WHITE_BRUSH);
    classejan.lpszMenuName = szNomeAplic;
    classejan.lpszClassName = szNomeAplic;
    if (!RegisterClass (&classejan)) return 0;
}
```

```

hjan = CreateWindow (szNomeAplic, "Programa Exemplo do Curso de C para Windows",
                    WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP, NULL, hCopia, NULL);

hCop = hCopia;
hAccel = LoadAccelerators (hCopia, szNomeAplic);
SetTimer (hjan, ID_TEMPORIZ, 50, NULL);
InitToolBar ();
InitCommonControls();

tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
                        IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
                        0, 16, 16, sizeof(TBBUTTON));

ShowWindow (hjan, iCmdMostrar);
UpdateWindow (hjan);

while (GetMessage (&msg, NULL, 0, 0)) {
    if (!TranslateAccelerator(hjan, hAccel, &msg)){
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}
return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    int xmouse, ymouse;
    int conta, cabre, ibuff;
    LPTOOLTIPTTEXT TTtext;
    char buff[20];
    static char fn[256];
    char *filefilter[] = { "Arquivos TPC (*.TPC)", "*.tpc", "Todos os Arquivos (*.*)", "*.*", "" };

    switch (iMsg) {
        case WM_CREATE:
            InicializaArq (hjan);
            lstrcpy (fn, (LPSTR) (((LPCREATESTRUCT) lParam)->lpCreateParams));

            if (lstrlen (fn) > 0) {
                GetFileTitle (fn, filename, sizeof (filename));
            }

            PoeTitulo (hjan, filename);
            hMenu = GetMenu (hjan);
            Brush[0] = CreateSolidBrush (RGB(255,255,255));
            Brush[1] = CreateSolidBrush (RGB(255,0,255));
            Brush[2] = CreateSolidBrush (RGB(0,255,0));
            Brush[3] = CreateSolidBrush (RGB(123,234,0));
            Brush[4] = CreateSolidBrush (RGB(255,255,255));
            Brush[5] = CreateSolidBrush (RGB(255,0,0));
            pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));
            pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));
            pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));
            esquema[0] = RGB(255,255,255);
            esquema[1] = RGB(255,0,255);
            esquema[2] = RGB(0,255,0);
            esquema[3] = RGB(73,189,102);
            esquema[4] = RGB(165,19,25);
            esquema[5] = RGB(12,138,2);
            hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");
            return 0;

        case WM_NOTIFY:
            TTtext = (LPTOOLTIPTTEXT) lParam;
            if (TTtext->hdr.code == TTN_NEEDTEXT)
                switch (TTtext->hdr.idFrom){
                    case IDM_GE: TTtext->lpszText = "Girar Esteira";
                        break;
                    case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
                        break;
                }
    }
}

```

```

        case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
            break;
        case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
            break;
        case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
            break;
    }

    return 0;
case WM_KEYDOWN:
    switch (wParam){
        case VK_HOME:
            xPontolns = 0;
            break;
        case VK_END:
            if(yPontolns == nlin)
                xPontolns = strlen(buff)%50;
            else xPontolns = 50;
            break;
        case VK_LEFT:
            xPontolns = xPontolns--;
            if((xPontolns == -1) && (yPontolns > 0)){
                xPontolns = 50;
                yPontolns--;
            }
            else if((xPontolns == -1) && (yPontolns == 0)) xPontolns = 0;
            break;
        case VK_RIGHT:
            xPontolns = xPontolns++;
            if ((xPontolns == 51) && (yPontolns < nlin)){
                xPontolns = 0;
                yPontolns++;
            }
            else if((yPontolns == nlin) && (xPontolns > strlen(buff)%50))
                xPontolns = strlen(buff)%50;
            break;
        case VK_UP:
            if (yPontolns > 0) yPontolns--;
            break;
        case VK_DOWN:
            if (yPontolns < nlin) yPontolns++;
            if((xPontolns > strlen(buff)%50) && (yPontolns == nlin))
                xPontolns = strlen(buff)%50;
            break;
        case VK_DELETE:
            for (conta = yPontolns*50 + xPontolns; conta < strlen(buff); conta++){
                buff[conta] = buff[conta + 1];
                buff[strlen(buff)] = '\0';
            }
            break;
        case VK_BACK:
            if (xPontolns > 0){
                xPontolns--;
                SendMessage(hjan, WM_KEYDOWN, VK_DELETE, 1L);
            }
            break;
        case VK_TAB:
            do{
                SendMessage(hjan, WM_CHAR, '\t', 1L);
            }while (xPontolns % 4 != 0);
            break;
        case VK_RETURN:
            for(conta = yPontolns*50 + xPontolns; conta < (yPontolns+1)*50; conta++){
                buff[conta + (yPontolns+1)*50 - 1] = buff[conta];
                buff[conta] = '\n';
            }
            xPontolns = 0;
            yPontolns++;
            if (yPontolns == 7){
                yPontolns = 6;
                MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
                    MB_ICONEXCLAMATION);
            }
    }
}

```

```

        }
        break;
    }
    InvalidateRect(hjan, NULL, TRUE);
    return 0;
case WM_CHAR:
    if (strlen(buff) > 299){
        MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        break;
    }
    else if((!isalnum(wParam))||(!isalnum(wParam)) && ((char)wParam != '\b') && ((char)wParam != '\t')){
        for (conta = strlen(buff); conta > (yPontolns * 50 + xPontolns); conta--){
            buff[conta] = buff[conta - 1];
        }
        buff[yPontolns * 50 + xPontolns] = (char) wParam;
        buff[strlen(buff) + 1] = '\0';
        xPontolns++;
        if(xPontolns == 50){
            xPontolns = 0;
            yPontolns++;
            if (yPontolns == 7){
                yPontolns = 6;
                MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
                    MB_ICONEXCLAMATION);
                InvalidateRect (hjan, NULL, TRUE);
            }
        }
    }
    nlin = strlen(buff)/50;
    InvalidateRect (hjan, NULL, FALSE);
    return 0;
case WM_COMMAND:
    switch (LOWORD(wParam)) {
        case IDM_ABRIR:
            if (modif)
                if (IDCANCEL == PerguntaSeSalva (hjan, filename))
                    return 0;

            memset(&fname, 0, sizeof(OPENFILENAME));
            fname.lStructSize = sizeof(OPENFILENAME);
            fname.hwndOwner = hjan;
            fname.lpstrFilter = filefilter[0];
            fname.nFilterIndex = 1;
            fname.lpstrFile = fn;
            fname.nMaxFile = sizeof(fn);
            fname.lpstrFileTitle = filename;
            fname.nMaxFileTitle = sizeof(filename)-1;
            fname.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

            if(!GetOpenFileName(&fname))
                break;

            if((fp=fopen(fn, "rb"))==NULL) {
                MessageBox(hjan, fn, "Não pode abrir o Arquivo", MB_OK);
                break;
            }

            numpecas = 0;

            strcpy(buff, "");
            fread(&numpecas, sizeof(int), 1, fp);
            fread(&wSelection, sizeof(int), 1, fp);
            fread(&xPontolns, sizeof(int), 1, fp);
            fread(&yPontolns, sizeof(int), 1, fp);
            fread(&cabre, sizeof(int), 1, fp);
            for(conta = 0; conta < cabre; conta++){
                fread(&ibuff, sizeof(int), 1, fp);
                buff[conta] = (char)ibuff;
            }
            buff[conta] = '\0';
            fclose(fp);
            SendMessage(hjan, WM_COMMAND, wSelection, 1L);
            modif = FALSE;
            InvalidateRect(hjan, NULL, TRUE);

```

```

        break;
    case IDM_SALVAR:
        memset(&fname, 0, sizeof(OPENFILENAME));
        fname.lStructSize = sizeof(OPENFILENAME);
        fname.hwndOwner = hjan;
        fname.lpstrFilter = filefilter[0];
        fname.nFilterIndex = 1;
        fname.lpstrFile = fn;
        fname.nMaxFile = sizeof(fn);
        fname.lpstrFileTitle = filename;
        fname.nMaxFileTitle = sizeof(filename)-1;
        fname.Flags = OFN_HIDEREADONLY;

        if(!GetSaveFileName(&fname))
            break;

        if((fp=fopen(fn, "wb"))==NULL) {
            MessageBox(hjan, fn, "Não pode salvar o Arquivo", MB_OK);
            break;
        }

        fwrite(&numpecas, sizeof(int), 1, fp);
        fwrite(&wSelection, sizeof(int), 1, fp);
        fwrite(&xPontosIns, sizeof(int), 1, fp);
        fwrite(&yPontosIns, sizeof(int), 1, fp);
        conta = strlen(buff);
        fwrite(&conta, sizeof(int), 1, fp);
        for(conta = 0; conta < strlen(buff); conta++){
            ibuff = (int)buff[conta];
            fwrite(&ibuff, sizeof(int), 1, fp);
        }
        fclose(fp);
        modif = FALSE;
        PoeTitulo(hjan, filename);
        InvalidateRect(hjan, NULL, TRUE);
        break;

    case IDM_IMPRIMIR:
        MessageBox(hjan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        break;

    case IDM_GE:
        if(xest < 97)
            ge = TRUE;
        break;

    case IDM_PP:
        if (!brcpeca && (xest > 23)){
            pp = TRUE;
            fr = 0;
            pc = FALSE;
        }
        break;

    case IDM_GB:
        if (brcpeca)
            gb = TRUE;
        break;

    case IDM_SB:
        if (brcpeca && !gc && !gb && !pp){
            numpecas++;
            modif = TRUE;
            brcpeca = FALSE;
            fr = 0;
            MessageBeep(0xFFFFFFFF);
            InvalidateRect(hjan, NULL, TRUE);
        }
        break;

    case IDM_PT:
        sprintf(buf, "Número de Peças Trabalhadas: %d", numpecas);
        MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        break;

    case IDM_E1:
    case IDM_E2:
    case IDM_E3:

```





```

        y2+=2;
        y4+=2;
    }
    if(x1 > 100){
        x1-=4;
        x4-=4;
    }
    if(y1 < 400){
        y1+=2;
        y4+=2;
    }
    if ((x3 - x2) != 0)
        fi3 = atan((y3 - y2)/(x3 - x2));
    else fi3 = pi/2;
    if ((x2 - x1) != 0)
        fi2 = atan((y2 - y1)/(x2 - x1));
    else fi2 = pi/2;
    if ((x2 <= 180) && (y2 >= 300) && (x1 <= 100) && (y1 >= 400)){
        pc = FALSE;
        pp = FALSE;
    }
    InvalidateRect (hjan, NULL, TRUE);
}
if (!pp && !pc && mov){
    fr+=2;
    if (fr > 10) {
        mov = FALSE;
        brcepa = TRUE;
        movest = TRUE;
        xest = 23;
        npeca = TRUE;
    }
    InvalidateRect (hjan, NULL, TRUE);
}
if (gb && !gc){
    if(x2 < 250){
        x2+=2;
        x4+=2;
    }
    if(y2 > 300){
        y2-=2;
        y4-=2;
    }
    if(x1 < 250){
        x1+=2;
        x4+=2;
    }
    if(y1 > 190){
        y1-=4;
        y4-=4;
    }
    if ((x3 - x2) != 0)
        fi3 = atan((y3 - y2)/(x3 - x2));
    else fi3 = pi/2;
    if ((x2 - x1) != 0)
        fi2 = atan((y2 - y1)/(x2 - x1));
    else fi2 = pi/2;
    if ((x2 == 250) && (y2 == 300) && (x1 == 250) && (y1 <= 190))
        gc = TRUE;
    InvalidateRect (hjan, NULL, TRUE);
}
else if (gb && gc){
    if(x2 <= 350){
        x2+=2;
        x4+=2;
    }
    if(y2 <= 300){
        y2+=2;
        y4+=2;
    }
}

```

```

        if(x1 <= 450){
            x1+=4;
            x4+=4;
        }
        if(y1 <= 340){
            y1+=2;
            y4+=2;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 >= 350) && (y2 >= 300) && (x1 >= 450) && (y1 >= 340)){
            gc = FALSE;
            gb = FALSE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    if(brspeca && (xnp == 0) && (xest > 23)) npeca = TRUE;
    if(npeca && (xest > 23) && (xnp < 23)) {
        xnp++;
        ynp++;
        InvalidateRect(hjan, NULL, TRUE);
    }
    else if (xnp == 23) npeca = FALSE;
        if(xest == 23) {
            npeca = FALSE;
            movest = TRUE;
            xnp = 0;
            ynp = 407;
        }
    if (ge && movest){
        fi4 = fi4+pi/10;
        xest+=2;
        if (xest == 97) {
            movest = FALSE;
            ge = FALSE;
            npeca = TRUE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    return 0;

```

case WM\_LBUTTONDOWN:

```

    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        SendMessage(hjan, WM_COMMAND, IDM_GE, 0L);
    else if((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PP, 0L);
    else if((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_GB, 0L);
    else if((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_SB, 0L);
    else if((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PT, 0L);
    return 0;

```

case WM\_RBUTTONDOWNBLCLK:

```

    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82))
        SendMessage(hjan, WM_COMMAND, IDM_E1, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100))
        SendMessage(hjan, WM_COMMAND, IDM_E2, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 104) && (ymouse <= 118))
        SendMessage(hjan, WM_COMMAND, IDM_E3, 0L);
    return 0;

```

```

case WM_MOUSEMOVE:
    xmouse = LOWORD(IParam);
    ymouse = HIWORD(IParam);
    if(((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82)) || ((xmouse <= 614)
        && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100)) || ((xmouse <= 614) && (xmouse >=
        600) && (ymouse >= 104) && (ymouse <= 118))) {
        hCursor = LoadCursor(hCop, "sedcur2");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if((xmouse < 360) && (ymouse < 162) && (ymouse > 45)) {
        hCursor = LoadCursor(hCop, "sedcur1");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if (((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        || ((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        || ((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450)) ||
        ((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450)) ||
        (((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450)))) {
        hCursor = LoadCursor(hCop, "sedcur3");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else {
        hCursor = LoadCursor(NULL, IDC_ARROW);
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }

    return 0;

case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);
    SelectObject(hdc, hFonte);
    SelectObject(hdc, Brush[wSelection/10 - 6]);
    SelectObject(hdc, pen[wSelection/10 - 9]);
    SetBkColor(hdc, esquema[wSelection/10 - 9]);
    SetTextColor (hdc, esquema[wSelection/10 - 6]);
    display (hdc);
    DeleteObject(pen[wSelection/10 - 9]);
    DeleteObject(Brush[wSelection/10 - 6]);
    DeleteObject((HGDIOBJ) hFonte);
    EndPaint (hjan, &ps);
    return 0;

case WM_CLOSE:
    DestroyWindow (hjan);
    return 0;

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

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

void InitToolbar () {
    tbButtons[0].iBitmap = 0;
    tbButtons[0].idCommand = IDM_GE;
    tbButtons[0].fsState = TBSTATE_ENABLED;
    tbButtons[0].fsStyle = TBSTYLE_BUTTON;
    tbButtons[0].dwData = 0;
    tbButtons[0].iString = 0;

    tbButtons[1].iBitmap = 0;
    tbButtons[1].idCommand = 0;
    tbButtons[1].fsState = TBSTATE_ENABLED;
    tbButtons[1].fsStyle = TBSTYLE_SEP;
    tbButtons[1].dwData = 0;
    tbButtons[1].iString = 0;

    tbButtons[2].iBitmap = 1;
    tbButtons[2].idCommand = IDM_PP;
    tbButtons[2].fsState = TBSTATE_ENABLED;
    tbButtons[2].fsStyle = TBSTYLE_BUTTON;
    tbButtons[2].dwData = 0;

```

```

tbButtons[2].iString = 0;
tbButtons[3].iBitmap = 2;
tbButtons[3].idCommand = IDM_GB;
tbButtons[3].fsState = TBSTATE_ENABLED;
tbButtons[3].fsStyle = TBSTYLE_BUTTON;
tbButtons[3].dwData = 0;
tbButtons[3].iString = 0;
tbButtons[4].iBitmap = 3;
tbButtons[4].idCommand = IDM_SB;
tbButtons[4].fsState = TBSTATE_ENABLED;
tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0;
tbButtons[4].iString = 0;
tbButtons[5].iBitmap = 0;
tbButtons[5].idCommand = 0;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0;
tbButtons[5].iString = 0;
tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0;
tbButtons[6].iString = 0;
}

void display(HDC hdc){
int i, j, k, cont;
char buffer[100], b[2];

//desenha a esteira
j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);
if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);
if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

//desenha o braço robótico
Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}
for(i = 0; i < 2; i++){
    MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
}

```

```

        MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
        LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
        MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
        LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
        MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
        LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
    }
    for(i = 0; i < 6; i++){
        MoveToEx (hdc, x1 + 20 - i, y1, NULL);
        LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
        MoveToEx (hdc, x1 - 20 + i, y1, NULL);
        LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
    }
    if(brcpeca)
        Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);
    Ellipse (hdc, 225, 375, 275, 425);
    Ellipse (hdc, 228, 378, 272, 422);
    MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
    LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
    Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);
    Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
    MoveToEx (hdc, x2 - 25*cos (fi2), y2 - 25*sin(fi2), NULL);
    LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
    Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
    Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

//desenha buffer de saída
    for(cont = 0; cont < 3; cont++){
        Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);
        sprintf(buffer, "Número de Peças: %d", numpeças);
        TextOut(hdc, 370, 410, buffer, strlen(buffer));
    }

//Apresenta os esquemas
    TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
    TextOut(hdc, 600, 68, "1", 1);
    TextOut(hdc, 600, 86, "2", 1);
    TextOut(hdc, 600, 104, "3", 1);

//Apresenta o texto do lembrete
    for(i = 0; i < 5; i++){
        if(i%2 == 0){
            MoveToEx(hdc, i, 20 + i, NULL);
            LineTo(hdc, i, 165 - i);
            LineTo(hdc, 370 - i, 165 - i);
        }
        MoveToEx(hdc, i, 30 + i, NULL);
        LineTo(hdc, 370 - i, 30 + i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    i = 0;
    k = 0;
    TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
    for(cont = 0; cont < strlen(buff); cont++){
        b[0] = buff[cont];
        b[1] = '\0';
        TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
        i++;
        if(i == 50){
            i = 0;
            k++;
        }
    }
    MoveToEx (hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
    LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
    MoveToEx (hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
    LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);
}

void InicializaArq (HWND hjan) {
    static char *szFiltro[] = { "Arquivos TPC (*.TPC)", "*.tpc", "Todos os Arquivos (*.*)", "**.*", "" };
    fname.lStructSize = sizeof (OPENFILENAME) ;
    fname.hwndOwner = hjan ;
}

```

```

fname.hInstance      = NULL ;
fname.lpstrFilter     = szFiltro [0] ;
fname.lpstrCustomFilter = NULL ;
fname.nMaxCustFilter  = 0 ;
fname.nFilterIndex    = 0 ;
fname.lpstrFile       = NULL ;
fname.nMaxFile        = _MAX_PATH ;
fname.lpstrFileTitle   = NULL ;
fname.nMaxFileTitle   = _MAX_FNAME + _MAX_EXT ;
fname.lpstrInitialDir  = NULL ;
fname.lpstrTitle       = NULL ;
fname.Flags           = 0 ;
fname.nFileOffset      = 0 ;
fname.nFileExtension  = 0 ;
fname.lpstrDefExt      = "tpc" ;
fname.lCustData        = 0L ;
fname.lpfnHook         = NULL ;
fname.lpTemplateName  = NULL ;
}

void PoeTitulo (HWND hjan, char *filename) {
    char szTit [64 + _MAX_FNAME + _MAX_EXT] ;

    strcpy (szTit, szTitulo);
    strcat (szTit, " - ");
    strcat (szTit, filename [0] ? filename : SEMTITULO);

    SetWindowText (hjan, szTit);
}

BOOL PerguntaSeSalva (HWND hjan, char *filename) {
    char szBuffer [64 + _MAX_FNAME + _MAX_EXT] ;
    BOOL nReturn ;

    szBuffer[0] = '\0';
    strcpy (szBuffer, "Salvar as alterações feitas em ");
    strcat (szBuffer, filename);

    nReturn = MessageBox (hjan, szBuffer, szTitulo, MB_YESNOCANCEL | MB_ICONQUESTION);

    if (nReturn == IDYES)
        if (!SendMessage (hjan, WM_COMMAND, IDM_SALVAR, 0L))
            nReturn = IDCANCEL ;

    return nReturn ;
}

```

---

## SEDCURSO.H

---

```

#define IDM_ABRIR      0
#define IDM_SALVAR     10
#define IDM_IMPRIMIR   20
#define IDM_SAIR       30
#define IDM_GE         40
#define IDM_PP         50
#define IDM_GB         60
#define IDM_SB         70
#define IDM_PT         80
#define IDM_E1         90
#define IDM_E2        100
#define IDM_E3        110
#define IDM_MOSTRARB   120
#define IDM_ESCB       130
#define IDM_SOBRE      140

#define IDTB_BMP       300
#define IDM_TOOLBAR    200
#define NUMBUTTONS     7
#define pi             3.1415
#define ID_TEMPORIZ    1
#define SEMTITULO      "(Sem Título)"

```

```
#include "sedcurso.h"

IDTB_BMP BITMAP "toolbarsedcurso.bmp"

sedcurso MENU {
    POPUP "&Arquivo" {
        MENUITEM "&Abrir\t^A", IDM_ABRIR
        MENUITEM "&Salvar\t^S", IDM_SALVAR
        MENUITEM "&Imprimir\t^I", IDM_IMPRIMIR
        MENUITEM SEPARATOR
        MENUITEM "Sai\r\t^R", IDM_SAIR
    }
    POPUP "&Sistema" {
        MENUITEM "&Girar Esteira\t^G", IDM_GE
        POPUP "&Braço Robótico" {
            MENUITEM "&Pegar Peça\t^P", IDM_PP
            MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
            MENUITEM "Soltar Peça\t^O", IDM_SB
        }
        MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
    }
    POPUP "&Cores" {
        MENUITEM "Esquema &1\t^H", IDM_E1
        MENUITEM "Esquema &2\t^J", IDM_E2
        MENUITEM "Esquema &3\t^K", IDM_E3
    }
    POPUP "&Botões" {
        MENUITEM "&Mostrar\t^M", IDM_MOSTRARB
        MENUITEM SEPARATOR
        MENUITEM "Es&conder\t^C", IDM_ESCB
    }
    POPUP "&Ajuda" {
        MENUITEM "So&bres\t^F1", IDM_SOBRE
    }
}

sedcurso ACCELERATORS {
    "^A",    IDM_ABRIR
    "^S",    IDM_SALVAR
    "I",     IDM_IMPRIMIR, CONTROL
    "^R",    IDM_SAIR
    "^G",    IDM_GE
    "^P",    IDM_PP
    "^B",    IDM_GB
    "^O",    IDM_SB
    "^E",    IDM_PT
    "H",     IDM_E1, CONTROL
    "^J",    IDM_E2
    "^K",    IDM_E3
    "^M",    IDM_MOSTRARB
    "^C",    IDM_ESCB
    VK_F1,   IDM_SOBRE, VIRTKEY
}

sedcurso ICON "sedcurso.ico"

sedcur1 CURSOR "cur1.cur"
sedcur2 CURSOR "cur2.cur"
sedcur3 CURSOR "cur3.cur"
```

---

Esse programa pode abrir arquivos com terminação .TPC, criado exclusivamente para ele. A janela de abertura e gravação de arquivos são caixas de diálogo comuns do WINDOWS, isto é, caixas de diálogo pré-definidas do próprio WINDOWS. Qualquer modificação feita no arquivo, quer ele tenha sido aberto, ou seja um novo arquivo, o usuário pode gravar os dados para sua utilização posterior. Nesse programa, qualquer alteração no arquivo é percebida pelo programa e, se o usuário tiver modificado seu conteúdo e tentar criar um novo arquivo, ou abrir um outro arquivo pré-



existente, ou mesmo sair do programa, uma caixa de mensagens (MessageBox) é apresentada perguntando se é desejado gravar as alterações. Esta caixa de mensagens é vista na Figura 7.

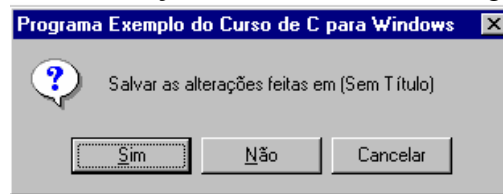


Figura 7

A janela de abertura de arquivos é mostrada na Figura 8 e a de gravação na Figura 9.

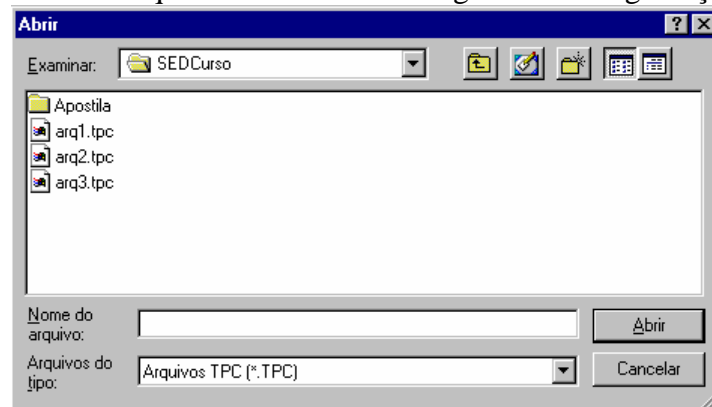


Figura 8

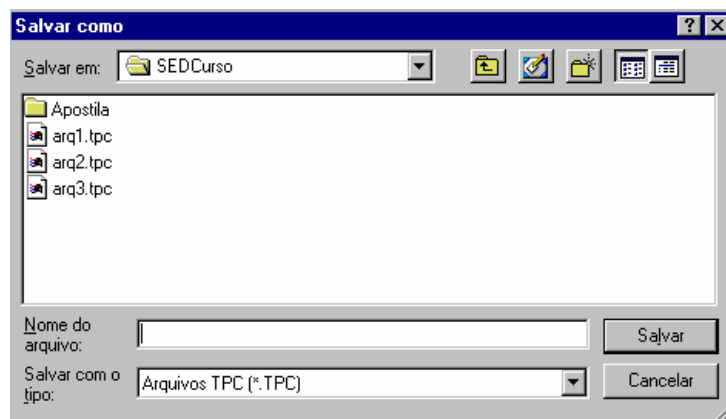


Figura 9

### Alterações no Programa SEDCURSO

Nesta Seção, a introdução da abertura e gravação de arquivos inclui as seguintes linhas de código no arquivo SEDCURSO.C:

➤ Protótipos de funções:

```
void InicializaArq (HWND);  
BOOL PerguntaSeSalva (HWND, char *);  
void PoeTitulo (HWND, char *);
```

que coloca o nome do arquivo iniciado como '(Sem Título)'; que pergunta se deseja salvar o arquivo modificado e que são funções que colocam o nome do arquivo a ser apresentado na barra de título da janela;, respectivamente.

➤ Variáveis globais:

```
OPENFILENAME fname;  
FILE *fp;  
char filename[64] = SEMTITULO;
```

estrutura de arquivo; ponteiro para a fila de bytes de arquivo e nome do arquivo com valor inicial '(Sem Título)' definido em SEDCURSO.H;

Na função WinMain, como pode ver o leitor, não há modificações para este programa. Para o procedimento de janela, tem-se como variáveis incluídas:

```
int conta, cabre, ibuff;
```

```
static char fn[256];
char *filefilter[] = { "Arquivos TPC (*.TPC)", "*.tpc", "Todos os Arquivos (*.*)", "**.*", "" };
```

que são variáveis para cálculo de número de bytes do buffer do lembrete; caminho do arquivo e o filtro que define os tipos de arquivos que podem ser abertos pelo programa (aparecem na janela de abertura).

Nas mensagens WINDOWS do programa, para este caso em que o programa deve colocar na barra de título da janela o nome do arquivo aberto, inclui-se em WM\_CREATE a inicialização do arquivo:

```
InicializaArq (hjan);
lstrcpy (fn, (LPSTR) (((LPCREATESTRUCT) lParam)->lCreateParams));
if (lstrlen (fn) > 0) {
    GetFileTitle (fn, filename, sizeof (filename));
}
```

```
PoeTitulo (hjan, filename);
```

que chama a função de inicialização de arquivos (vista mais adiante); define o caminho inicial para o arquivo e coloca o nome do arquivo na barra de título, respectivamente.

Na mensagem WM\_COMMAND, os itens de menu IDM\_ABRIR e IDM\_SALVAR são utilizados para abrir a caixa de diálogo comum específica de cada tipo. Isto é, a caixa de diálogo comum para abrir e a caixa de diálogo comum para salvar.

IDM\_ABRIR – se é desejado abrir um arquivo, avalia-se se existe alguma modificação para saber se o usuário quer gravar:

```
if (modif)
    if (IDCANCEL == PerguntaSeSalva (hjan, filename))
        return 0 ;
```

aloca memória para fname:

```
memset(&fname, 0, sizeof(OPENFILENAME));
```

inicializa a estrutura OPENFILENAME e abre a caixa de diálogo comum Abrir:

```
fname.lStructSize = sizeof(OPENFILENAME); - especifica o tamanho da
estrutura;
fname.hwndOwner = hjan; - identifica qual programa está chamando esta caixa de
diálogo;
fname.lpstrFilter = filefilter[0]; - identifica os tipos de arquivos que devem ser
apresentados na caixa de diálogo;
fname.nFilterIndex = 1; - especifica o primeiro par de tipos de arquivos a serem
mostrados;
fname.lpstrFile = fn; - especifica o caminho inicial do arquivo a ser aberto;
fname.nMaxFile = sizeof(fn); - especifica o tamanho do caminho em caracteres;
fname.lpstrFileTitle = filename; - ponteiro para o nome do arquivo que recebe a
extensão especificada;
fname.nMaxFileTitle = sizeof(filename)-1; - especifica o tamanho do nome do
arquivo;
fname.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY; -
identificadores para inicialização da caixa de diálogo abrir: apenas arquivos existentes
podem ser solicitados para abertura e não habilita a caixa de seleção de “apenas
leitura”;
```

Esta estrutura é maior do que aqui se apresenta. Mas esses parâmetros são os mais necessários ao que se deseja na maioria dos programas.

Com essas definições, a caixa de diálogo comum Abrir é apresentada:

```
if(!GetOpenFileName(&fname))
    break;
```

Para um arquivo inexistente no diretório, cujo nome foi digitado para ser aberto, uma mensagem é enviada, declarando que não é possível abri-lo:

```
if((fp=fopen(fn, "r"))==NULL) {
    MessageBox(hjan, fn, "Não pode abrir o Arquivo", MB_OK);
    break;
}
```

Todas as variáveis para receber os novos valores do arquivo que está sendo aberto são inicializadas:

```
numpecas = 0;
strcpy(buff, "");
```

Lê-se o número de peças trabalhadas, o esquema de cores, a localização do cursor de texto, o número de caracteres do texto do lembrete e o texto do lembrete, respectivamente:

```
fread(&numpecas, sizeof(int), 1, fp);
fread(&wSelection, sizeof(int), 1, fp);
fread(&xPontolns, sizeof(int), 1, fp);
fread(&yPontolns, sizeof(int), 1, fp);
fread(&cabre, sizeof(int), 1, fp);
for(conta = 0; conta < cabre; conta++){
    fread(&ibuff, sizeof(int), 1, fp);*
    buff[conta] = (char)ibuff;
}
```

fecha-se o vetor com o texto do lembrete e fecha-se o arquivo:

```
buff[conta] = '\0';
fclose(fp);
```

envia uma mensagem para definir o esquema de cores utilizado no arquivo:

```
SendMessage(hjan, WM_COMMAND, wSelection, 1L);
```

torna a variável `modif` falsa para afirmar que não há modificações no arquivo e repinta-se a janela:

```
modif = FALSE;
InvalidateRect(hjan, NULL, TRUE);
break;
```

**IDM\_GRAVAR** – processa a gravação de arquivo. A estruturação da gravação de arquivo é inversa a da abertura. A estrutura **OPENFILENAME** é inicializada apenas diferenciando a linha:

```
fname.Flags = OFN_HIDEREADONLY;
```

que determina que o usuário pode colocar qualquer nome. Cada byte do arquivo é gravado na mesma sequência que foi apresentada a abertura de arquivos (função `fwrite`).

O código da função de inicialização de arquivos é a própria estrutura **OPENFILENAME**. Nela são definidos todos os parâmetros necessários para os tipos de arquivos que podem ser abertos, tamanho máximo de nome, caminho, e outros. Para se inicializar arquivos utilizando procedimentos na forma desse programa, a estrutura pode ser inicializada sempre com os valores dos parâmetros como definidos nessa função, observando-se o tipo de arquivo inicial, dado pela linha

```
fname.lpstrDefExt = "tpc" ;
```

que aqui é **.TPC**. Também, deve-se saber que as macros **\_MAX\_PATH**, **\_MAX\_FNAME** e **\_MAX\_EXT** são definidas em **WINDOWS.H**.

Os valores iguais a **NULL** ou zero nas linhas:

```
fname.nFilterIndex          = 0;
fname.lpstrFile              = NULL;
fname.lpstrFileTitle         = NULL;
fname.Flags                  = 0;
```

são modificados quando a estrutura é inicializada na seleção do menu **Abrir** ou na seleção **Gravar**, de acordo com o que se necessita.

A função que modifica o título do programa para colocar o nome do arquivo é processada como a seguir: recebe o identificador da janela e o nome do arquivo

```
void PoeTitulo (HWND hjan, char *filename)
```

cria uma variável que contém um número máximo de caracteres para ser posto na barra de títulos:

```
char szTit [64 + _MAX_FNAME + _MAX_EXT] ;
```

copia para essa variável **szTitulo**:

```
strcpy (szTit, szTitulo);
```

coloca um caractere para separação e o nome do arquivo:

```
strcat (szTit, " - ");
strcat (szTit, filename [0] ? filename : SEMTITULO);
```

Caso o arquivo ainda não tenha título, é colocado o texto **“(Sem Título)”** definido na constante **SEMTITULO**.

Coloca o novo título da barra de título do programa:

```
SetWindowText (hjan, szTit);
```

Por fim, a última função do programa funciona como a seguir: recebe o identificador da janela e o título do arquivo:

```
BOOL PerguntaSeSalva (HWND hjan, char *filename)
```

cria uma variável que contém um número máximo de caracteres para ser apresentado na caixa de mensagem e uma variável para ser devolvida à janela principal do programa:

```
char szBuffer [64 + _MAX_FNAME + _MAX_EXT] ;
```

```
BOOL nReturn ;
```

A variável é inicializada colocando o texto “Salvar as alterações feitas em” com o nome do arquivo:

```
szBuffer[0] = '\0';
```

```
strcpy (szBuffer, "Salvar as alterações feitas em ");
```

```
strcat (szBuffer, filename);
```

determina o valor da variável `nReturn`, de acordo com a resposta dada à caixa de mensagem:

```
nReturn = MessageBox (hjan, szBuffer, szTitulo, MB_YESNOCANCEL | MB_ICONQUESTION);
```

Se o usuário do programa pressionou **OK**, é feita uma chamada à caixa de diálogo comum de gravação. Se o usuário desistir de gravar, pressionando o botão **Cancelar**, torna o valor da variável `nReturn = IDCANCEL`, retornando o programa à última tarefa executada.

```
if (nReturn == IDYES)
```

```
if (!SendMessage (hjan, WM_COMMAND, IDM_GRAVAR, 0L))
```

```
nReturn = IDCANCEL ;
```

```
return nReturn ;
```

O arquivo `SEDCURSO.H` inclui as linhas

```
#define SEMTITULO "(Sem Título)"
```

que define a constante utilizada no programa: `SEMTITULO` que contém o valor de inicialização de arquivos.

## 10. UTILIZANDO A IMPRESSORA

Para utilizar a impressora, são necessários alguns recursos específicos para determinar o formato do papel, qual impressora deve imprimir o trabalho, entre outros.

Quando se deseja imprimir algum trabalho desenvolvido em um aplicativo **WINDOWS**, utiliza-se uma caixa de diálogo comum, onde são definidos todos os dados necessários, como os citados anteriormente e outros.

A impressão é feita de forma similar a apresentação na janela, seja texto ou gráfico. Isto é, as funções para imprimir texto e gráfico são as mesmas utilizadas para apresentá-los na janela. Entretanto, torna-se necessário obter da impressora as dimensões específicas do tamanho do papel, calcular todas as coordenadas do que vai ser impresso no papel, bem como número de páginas, caracteres por linha, linhas por página, tamanho da fonte, entre outros.

Na impressão, é gerada uma imagem para cada página de tudo o que for colocado para ser impresso. Dessa forma, o código deve avaliar em que página está sendo impresso o texto ou gráfico específico daquela localização.

Também utiliza-se frequentemente uma caixa de diálogo para cancelar a impressão, onde antes do trabalho ser enviado completamente, o usuário pode desistir de imprimi-lo.

-----  
SEDCURSO.C  
-----

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

```
#include <commctrl.h>
```

```
#include <math.h>
```

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
#include <stdlib.h>
```

```
#include "sedcurso.h"
```

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

```
void InitToolbar (void);
```

```
void display(HDC);
```

```
void InicializaArq (HWND);
```

```
BOOL PerguntaSeSalva (HWND, char *);
```

```
void PoeTitulo (HWND, char *);
```

```
BOOL CALLBACK ProcAborto (HDC, int);
```

\*

```

BOOL ImprimirArquivo (HINSTANCE, HWND, char *);
HDC ObtemCDImpress (void);

HINSTANCE hCop;
HBRUSH Brush[6];
HPEN pen[3];
HCURSOR hCursor;
COLORREF esquema[6];
TBBUTTON tbButtons [NUMBUTTONS];
HWND tbwnd;
int nlin = 0, xPontols = 0, yPontols = 0, ToolBarActive, xest = 23, fr = 0, numpecas = 0, x1 = 280, x2 = 260, x3 = 250,
x4 = 100, x5 = 25, x6 = 120, y1 = 190, y2 = 300, y3 = 400, y4 = 320, y5 = 475, xnp = 0, ynp = 407;
UINT wSelection = IDM_E1;
HMENU hMenu;
HFONT hFonte;
double fi1 = 0.0, fi2 = 1.5707, fi3 = 1.5707, fi4 = 1.5707;
BOOL gc = FALSE, pc = FALSE, ge = FALSE, pp = FALSE, gb = FALSE, sb = FALSE, pt = FALSE, mov = FALSE,
movest = TRUE, brcepa = FALSE, npeca = FALSE, modif = FALSE;
OPENFILENAME fname;
FILE *fp;
char filename[64] = SEMTITULO;
char buff[300], lembrete[] = "Lembrete!";
char szTitulo[] = "Programa Exemplo do Curso de C para Windows";
HWND hDlgImpressao;
BOOL bSucesso, bUsuarioAborta;

char szNomeAplic[] = "sedcurso";

int WINAPI WinMain (HINSTANCE hCopia, HINSTANCE hCopiaAnt, LPSTR szLinhaCmd, int iCmdMostrar) {
    HWND hjan;
    MSG msg;
    WNDCLASS classejan;
    HACCEL hAccel;

    classejan.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    classejan.lpfnWndProc = ProcJan;
    classejan.cbClsExtra = 0;
    classejan.cbWndExtra = 0;
    classejan.hInstance = hCopia;
    classejan.hIcon = LoadIcon (hCopia, szNomeAplic);
    classejan.hCursor = LoadCursor (NULL, IDC_ARROW);
    classejan.hbrBackground = GetStockObject (WHITE_BRUSH);
    classejan.lpszMenuName = szNomeAplic;
    classejan.lpszClassName = szNomeAplic;

    if (!RegisterClass (&classejan)) return 0;

    hjan = CreateWindow (szNomeAplic, "Programa Exemplo do Curso de C para Windows",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, HWND_DESKTOP, NULL, hCopia, NULL);

    hCop = hCopia;
    hAccel = LoadAccelerators (hCopia, szNomeAplic);
    SetTimer (hjan, ID_TEMPORIZ, 50, NULL);
    InitToolBar ();
    InitCommonControls();

    tbwnd = CreateToolBarEx (hjan, WS_VISIBLE | WS_CHILD | WS_BORDER | TBSTYLE_TOOLTIPS,
        IDM_TOOLBAR, NUMBUTTONS, hCopia, IDTB_BMP, tbButtons, NUMBUTTONS, 0,
        0, 16, 16, sizeof(TBBUTTON));

    ShowWindow (hjan, iCmdMostrar);
    UpdateWindow (hjan);

    while (GetMessage (&msg, NULL, 0, 0)) {
        if (!TranslateAccelerator(hjan, hAccel, &msg)){
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }
    return msg.wParam;
}

LRESULT CALLBACK ProcJan (HWND hjan, UINT iMsg, WPARAM wParam, LPARAM lParam){
    HDC hdc;

```

```

PAINTSTRUCT ps;
RECT rect;
int xmouse, ymouse;
int conta, cabre, ibuff;
LPTOOLTIPTTEXT TTtext;
char buff[20];
static char fn[256];
char *filefilter[] = { "Arquivos TPC (*.TPC)", "*.tpc", "Todos os Arquivos (*.*)", "*.*", "" };

switch (iMsg) {
case WM_CREATE:
    InicializaArq (hjan);
    lstrcpy (fn, (LPSTR) (((LPCREATESTRUCT) lParam)->lCreateParams));
    if (lstrlen (fn) > 0) {
        GetFileTitle (fn, filename, sizeof (filename));
    }

    PoeTitulo (hjan, filename);
    hMenu = GetMenu (hjan);
    Brush[0] = CreateSolidBrush (RGB(255,255,255));
    Brush[1] = CreateSolidBrush (RGB(255,0,255));
    Brush[2] = CreateSolidBrush (RGB(0,255,0));
    Brush[3] = CreateSolidBrush (RGB(123,234,0));
    Brush[4] = CreateSolidBrush (RGB(255,255,255));
    Brush[5] = CreateSolidBrush (RGB(255,0,0));
    pen[0] = CreatePen (PS_SOLID, 1, RGB(34,25,155));
    pen[1] = CreatePen (PS_SOLID, 1, RGB(255,25,0));
    pen[2] = CreatePen (PS_SOLID, 1, RGB(65,89,205));
    esquema[0] = RGB(255,255,255);
    esquema[1] = RGB(255,0,255);
    esquema[2] = RGB(0,255,0);
    esquema[3] = RGB(73,189,102);
    esquema[4] = RGB(165,19,25);
    esquema[5] = RGB(12,138,2);
    hFonte = CreateFont(14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Times New Roman");
    return 0;

case WM_NOTIFY:
    TTtext = (LPTOOLTIPTTEXT) lParam;
    if (TTtext->hdr.code == TTN_NEEDTEXT)
        switch (TTtext->hdr.idFrom){
            case IDM_GE: TTtext->lpszText = "Girar Esteira";
                break;
            case IDM_PP: TTtext->lpszText = "Braço Robótico Pegar Peça";
                break;
            case IDM_GB: TTtext->lpszText = "Girar Braço Robótico";
                break;
            case IDM_SB: TTtext->lpszText = "Braço Robótico Soltar Peça";
                break;
            case IDM_PT: TTtext->lpszText = "Contar Peças Trabalhadas";
                break;
        }

    return 0;

case WM_KEYDOWN:
    switch (wParam){
        case VK_HOME:
            xPontolns = 0;
            break;
        case VK_END:
            if(yPontolns == nlin)
                xPontolns = strlen(buff)%50;
            else xPontolns = 50;
            break;
        case VK_LEFT:
            xPontolns = xPontolns--;
            if((xPontolns == -1) && (yPontolns > 0)){
                xPontolns = 50;
                yPontolns--;
            }
            else if((xPontolns == -1) && (yPontolns == 0)) xPontolns = 0;
            break;
    }
}

```

```

case VK_RIGHT:
    xPontols = xPontols++;
    if ((xPontols == 51) && (yPontols < nlin)){
        xPontols = 0;
        yPontols++;
    }
    else if((yPontols == nlin) && (xPontols > strlen(buff)%50))
        xPontols = strlen(buff)%50;
    break;
case VK_UP:
    if (yPontols > 0) yPontols--;
    break;
case VK_DOWN:
    if (yPontols < nlin) yPontols++;
    if((xPontols > strlen(buff)%50) && (yPontols == nlin))
        xPontols = strlen(buff)%50;
    break;
case VK_DELETE:
    for (conta = yPontols*50 + xPontols; conta < strlen(buff); conta++){
        buff[conta] = buff[conta + 1];
        buff[strlen(buff)] = '\0';
    }
    break;
case VK_BACK:
    if (xPontols > 0){
        xPontols--;
        SendMessage (hjan, WM_KEYDOWN, VK_DELETE, 1L);
    }
    break;
case VK_TAB:
    do{
        SendMessage (hjan, WM_CHAR, '\t', 1L);
    }while (xPontols % 4 != 0);
    break;
case VK_RETURN:
    for(conta = yPontols*50 + xPontols; conta < (yPontols+1)*50; conta++){
        buff[conta + (yPontols+1)*50 - 1] = buff[conta];
        buff[conta] = ' ';
    }
    xPontols = 0;
    yPontols++;
    if (yPontols == 7){
        yPontols = 6;
        MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
            MB_ICONEXCLAMATION);
    }
    break;
}
InvalidateRect(hjan, NULL, TRUE);
return 0;

case WM_CHAR:
    if (strlen(buff) > 299){
        MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        break;
    }
    else if((!isalnum(wParam)) || (!isalnum(wParam)) && ((char)wParam != '\b') && ((char)wParam != '\t')){
        for (conta = strlen(buff); conta > (yPontols * 50 + xPontols); conta--){
            buff[conta] = buff[conta - 1];
        }
        buff[yPontols * 50 + xPontols] = (char) wParam;
        buff[strlen(buff) + 1] = '\0';
        xPontols++;
        if(xPontols == 50){
            xPontols = 0;
            yPontols++;
            if (yPontols == 7){
                yPontols = 6;
                MessageBox(hjan, "Não pode colocar mais texto!", szNomeAplic, MB_OK |
                    MB_ICONEXCLAMATION);
                InvalidateRect (hjan, NULL, TRUE);
            }
        }
    }
}

```

```

    }
    nlin = strlen(buff)/50;
    InvalidateRect (hjan, NULL, FALSE);
    return 0;

case WM_COMMAND:
    switch (LOWORD(wParam)) {
        case IDM_ABRIR:
            if (modif)
                if (IDCANCEL == PerguntaSeSalva (hjan, filename))
                    return 0;

            memset(&fname, 0, sizeof(OPENFILENAME));
            fname.lStructSize = sizeof(OPENFILENAME);
            fname.hwndOwner = hjan;
            fname.lpstrFilter = filefilter[0];
            fname.nFilterIndex = 1;
            fname.lpstrFile = fn;
            fname.nMaxFile = sizeof(fn);
            fname.lpstrFileTitle = filename;
            fname.nMaxFileTitle = sizeof(filename)-1;
            fname.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

            if(!GetOpenFileName(&fname))
                break;

            if((fp=fopen(fn, "rb"))==NULL) {
                MessageBox(hjan, fn, "Não pode abrir o Arquivo", MB_OK);
                break;
            }

            numpecas = 0;
            strcpy(buff, "");
            fread(&numpecas, sizeof(int), 1, fp);
            fread(&wSelection, sizeof(int), 1, fp);
            fread(&xPontosIns, sizeof(int), 1, fp);
            fread(&yPontosIns, sizeof(int), 1, fp);
            fread(&cabre, sizeof(int), 1, fp);
            for(conta = 0; conta < cabre; conta++){
                fread(&ibuff, sizeof(int), 1, fp);
                buff[conta] = (char)ibuff;
            }
            buff[conta] = '\0';
            fclose(fp);
            SendMessage(hjan, WM_COMMAND, wSelection, 1L);
            modif = FALSE;
            InvalidateRect(hjan, NULL, TRUE);
            break;

        case IDM_SALVAR:
            memset(&fname, 0, sizeof(OPENFILENAME));
            fname.lStructSize = sizeof(OPENFILENAME);
            fname.hwndOwner = hjan;
            fname.lpstrFilter = filefilter[0];
            fname.nFilterIndex = 1;
            fname.lpstrFile = fn;
            fname.nMaxFile = sizeof(fn);
            fname.lpstrFileTitle = filename;
            fname.nMaxFileTitle = sizeof(filename)-1;
            fname.Flags = OFN_HIDEREADONLY;

            if(!GetSaveFileName(&fname))
                break;

            if((fp=fopen(fn, "wb"))==NULL) {
                MessageBox(hjan, fn, "Não pode salvar o Arquivo", MB_OK);
                break;
            }

            fwrite(&numpecas, sizeof(int), 1, fp);
            fwrite(&wSelection, sizeof(int), 1, fp);
            fwrite(&xPontosIns, sizeof(int), 1, fp);
            fwrite(&yPontosIns, sizeof(int), 1, fp);
            conta = strlen(buff);
            fwrite(&conta, sizeof(int), 1, fp);
            for(conta = 0; conta < strlen(buff); conta++){

```



```

        ibuff = (int)buff[conta];
        fwrite(&ibuff, sizeof(int), 1, fp);
    }
    fclose(fp);
    modif = FALSE;
    PoeTitulo(hjan, filename);
    InvalidateRect(hjan, NULL, TRUE);
    break;
case IDM_IMPRIMIR:
    if(ImprimirArquivo(hCop, hjan, filename))
        MessageBox(hjan, "Não pôde imprimir!", szNomeAplic, MB_OK |
            MB_ICONEXCLAMATION);
    break;
case IDM_GE:
    if(xest < 97)
        ge = TRUE;
    break;
case IDM_PP:
    if(!brcpeca && (xest > 23)){
        pp = TRUE;
        fr = 0;
        pc = FALSE;
    }
    break;
case IDM_GB:
    if(brcpeca)
        gb = TRUE;
    break;
case IDM_SB:
    if(brcpeca && !gc && !gb && !pp){
        numpecas++;
        modif = TRUE;
        brcpeca = FALSE;
        fr = 0;
        MessageBeep(0xFFFFFFFF);
        InvalidateRect(hjan, NULL, TRUE);
    }
    break;
case IDM_PT:
    sprintf(buf, "Número de Peças Trabalhadas: %d", numpecas);
    MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
    break;
case IDM_E1:
case IDM_E2:
case IDM_E3:
    hMenu = GetMenu(hjan);
    CheckMenuItem(hMenu, wSelection, MF_UNCHECKED);
    wSelection = LOWORD(wParam);
    CheckMenuItem(hMenu, wSelection, MF_CHECKED);
    DeleteObject((HGDIOBJ) SetClassLong(hjan, GCL_HBRBACKGROUND, (LONG)
        Brush[wSelection/10 - 9]));
    InvalidateRect(hjan, NULL, TRUE);
    sprintf(buf, "Esquema de Cores %d", wSelection/10 - 8);
    MessageBox(hjan, buf, szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
    break;
case IDM_SAIR:
    if(IDOK == MessageBox(hjan, "Você deseja realmente terminar o programa?",
        szNomeAplic, MB_OKCANCEL | MB_ICONEXCLAMATION))
        SendMessage(hjan, WM_CLOSE, 0, 0L);
    break;
case IDM_MOSTRARB:
    ToolBarActive = 1;
    ShowWindow(tbwnd, SW_RESTORE);
    InvalidateRect(hjan, NULL, 1);
    break;
case IDM_ESCB:
    ToolBarActive = 0;
    ShowWindow(tbwnd, SW_HIDE);
    InvalidateRect(hjan, NULL, 1);

```

```

        break;
    case IDM_SOBRE:
        MessageBox(hjan, "Programa de Simulação de Sistema\npara Aprendizado de
        Programação\nem C para WINDOWS!\n\nEduard Montgomery Meira Costa,
        DSc\n\n\t\t\t2002", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
        return 0;
    }
    return 0;

case WM_TIMER:
    if (pp && !pc){
        mov = TRUE;
        if(x2 > 190){
            x2-=2;
            x4-=2;
        }
        if(y2 > 300){
            y2-=4;
            y4-=4;
        }
        if(x1 > 140){
            x1-=4;
            x4-=4;
        }
        if(y1 > 190){
            y1-=4;
            y4-=4;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 <= 250) && (y2 <= 300) && (x1 <= 250) && (y1 <= 190))
            pc = TRUE;
        InvalidateRect(hjan, NULL, TRUE);
    }
    else if (pp && pc){
        mov = TRUE;
        if(x2 > 180){
            x2-=2;
            x4-=2;
        }
        if(y2 < 300){
            y2+=2;
            y4+=2;
        }
        if(x1 > 100){
            x1-=4;
            x4-=4;
        }
        if(y1 < 400){
            y1+=2;
            y4+=2;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 <= 180) && (y2 >= 300) && (x1 <= 100) && (y1 >= 400)){
            pc = FALSE;
            pp = FALSE;
        }
        InvalidateRect(hjan, NULL, TRUE);
    }
    if (!pp && !pc && mov){
        fr+=2;
    }

```

```

        if (fr > 10) {
            mov = FALSE;
            brcpeca = TRUE;
            movest = TRUE;
            xest = 23;
            npeca = TRUE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    if (gb && !gc){
        if(x2 < 250){
            x2+=2;
            x4+=2;
        }
        if(y2 > 300){
            y2-=2;
            y4-=2;
        }
        if(x1 < 250){
            x1+=2;
            x4+=2;
        }
        if(y1 > 190){
            y1-=4;
            y4-=4;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 == 250) && (y2 == 300) && (x1 == 250) && (y1 <= 190))
            gc = TRUE;
        InvalidateRect (hjan, NULL, TRUE);
    }
    else if (gb && gc){
        if(x2 <= 350){
            x2+=2;
            x4+=2;
        }
        if(y2 <= 300){
            y2+=2;
            y4+=2;
        }
        if(x1 <= 450){
            x1+=4;
            x4+=4;
        }
        if(y1 <= 340){
            y1+=2;
            y4+=2;
        }
        if ((x3 - x2) != 0)
            fi3 = atan((y3 - y2)/(x3 - x2));
        else fi3 = pi/2;
        if ((x2 - x1) != 0)
            fi2 = atan((y2 - y1)/(x2 - x1));
        else fi2 = pi/2;
        if ((x2 >= 350) && (y2 >= 300) && (x1 >= 450) && (y1 >= 340)){
            gc = FALSE;
            gb = FALSE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    if(brcpeca && (xnp == 0) && (xest > 23)) npeca = TRUE;
    if(npeca && (xest > 23) && (xnp < 23)) {
        xnp++;
        ynp++;
        InvalidateRect(hjan, NULL, TRUE);
    }

```

```

    }
    else if (xnp == 23) npeca = FALSE;
        if(xest == 23) {
            npeca = FALSE;
            movest = TRUE;
            xnp = 0;
            ynp = 407;
        }
    if (ge && movest){
        fi4 = fi4+pi/10;
        xest+=2;
        if (xest == 97) {
            movest = FALSE;
            ge = FALSE;
            npeca = TRUE;
        }
        InvalidateRect (hjan, NULL, TRUE);
    }
    return 0;

case WM_LBUTTONDOWN:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        SendMessage(hjan, WM_COMMAND, IDM_GE, 0L);
    else if((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_PP, 0L);
    else if((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450))
        SendMessage(hjan, WM_COMMAND, IDM_GB, 0L);
    else if((xmouse<=300) && (xmouse>=270) && (ymouse>=400) && (ymouse<=450))
        SendMessage(hjan, WM_COMMAND, IDM_SB, 0L);
    else if((xmouse<=550)&&(xmouse>=350) && (ymouse>=400) && (ymouse<=450))
        SendMessage(hjan, WM_COMMAND, IDM_PT, 0L);

    return 0;

case WM_RBUTTONDOWNBLCLK:
    xmouse = LOWORD (lParam);
    ymouse = HIWORD (lParam);
    if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82))
        SendMessage(hjan, WM_COMMAND, IDM_E1, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100))
        SendMessage(hjan, WM_COMMAND, IDM_E2, 0L);
    else if((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 104) && (ymouse <= 118))
        SendMessage(hjan, WM_COMMAND, IDM_E3, 0L);

    return 0;

case WM_MOUSEMOVE:
    xmouse = LOWORD(lParam);
    ymouse = HIWORD(lParam);
    if(((xmouse <= 614) && (xmouse >= 600) && (ymouse >= 68) && (ymouse <= 82)) || ((xmouse <= 614)
        && (xmouse >= 600) && (ymouse >= 86) && (ymouse <= 100)) || ((xmouse <= 614) && (xmouse >=
        600) && (ymouse >= 104) && (ymouse <= 118))){
        hCursor = LoadCursor(hCop, "sedcur2");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if((xmouse < 360) && (ymouse < 162) && (ymouse > 45)){
        hCursor = LoadCursor(hCop, "sedcur1");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else if (((xmouse <= 145) && (xmouse >= 95) && (ymouse >= 450) && (ymouse <= 500))
        ||((xmouse <= 230) && (xmouse >= 200) && (ymouse >= 400) && (ymouse <= 450))
        ||((xmouse <= 265) && (xmouse >= 233) && (ymouse >= 400) && (ymouse <= 450)) ||
        ((xmouse <= 300) && (xmouse >= 270) && (ymouse >= 400) && (ymouse <= 450))
        ||((xmouse <= 550) && (xmouse >= 350) && (ymouse >= 400) && (ymouse <= 450))){
        hCursor = LoadCursor(hCop, "sedcur3");
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }
    else {
        hCursor = LoadCursor(NULL, IDC_ARROW);
        SetClassLong (hjan, GCL_HCURSOR, (LONG)hCursor);
    }

```

```

    }
    return 0;

case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);
    SelectObject(hdc, hFonte);
    SelectObject(hdc, Brush[wSelection/10 - 6]);
    SelectObject(hdc, pen[wSelection/10 - 9]);
    SetBkColor(hdc, esquema[wSelection/10 - 9]);
    SetTextColor (hdc, esquema[wSelection/10 - 6]);
    display (hdc);
    DeleteObject(pen[wSelection/10 - 9]);
    DeleteObject(Brush[wSelection/10 - 6]);
    DeleteObject((HGDIOBJ) hFonte);
    EndPaint (hjan, &ps);
    return 0;

case WM_CLOSE:
    DestroyWindow (hjan);
    return 0;

case WM_DESTROY:
    PostQuitMessage (0);
    return 0;
}
return DefWindowProc (hjan, iMsg, wParam, lParam);
}

void InitToolbar () {
tbButtons[0].iBitmap = 0;
tbButtons[0].idCommand = IDM_GE;
tbButtons[0].fsState = TBSTATE_ENABLED;
tbButtons[0].fsStyle = TBSTYLE_BUTTON;
tbButtons[0].dwData = 0;
tbButtons[0].iString = 0;

tbButtons[1].iBitmap = 0;
tbButtons[1].idCommand = 0;
tbButtons[1].fsState = TBSTATE_ENABLED;
tbButtons[1].fsStyle = TBSTYLE_SEP;
tbButtons[1].dwData = 0;
tbButtons[1].iString = 0;

tbButtons[2].iBitmap = 1;
tbButtons[2].idCommand = IDM_PP;
tbButtons[2].fsState = TBSTATE_ENABLED;
tbButtons[2].fsStyle = TBSTYLE_BUTTON;
tbButtons[2].dwData = 0;
tbButtons[2].iString = 0;

tbButtons[3].iBitmap = 2;
tbButtons[3].idCommand = IDM_GB;
tbButtons[3].fsState = TBSTATE_ENABLED;
tbButtons[3].fsStyle = TBSTYLE_BUTTON;
tbButtons[3].dwData = 0;
tbButtons[3].iString = 0;

tbButtons[4].iBitmap = 3;
tbButtons[4].idCommand = IDM_SB;
tbButtons[4].fsState = TBSTATE_ENABLED;
tbButtons[4].fsStyle = TBSTYLE_BUTTON;
tbButtons[4].dwData = 0;
tbButtons[4].iString = 0;

tbButtons[5].iBitmap = 0;
tbButtons[5].idCommand = 0;
tbButtons[5].fsState = TBSTATE_ENABLED;
tbButtons[5].fsStyle = TBSTYLE_SEP;
tbButtons[5].dwData = 0;
tbButtons[5].iString = 0;

tbButtons[6].iBitmap = 4;
tbButtons[6].idCommand = IDM_PT;
tbButtons[6].fsState = TBSTATE_ENABLED;
tbButtons[6].fsStyle = TBSTYLE_BUTTON;
tbButtons[6].dwData = 0;

```

```

tbButtons[6].iString = 0;
}

void display(HDC hdc){
int i, j, k, cont;
char buffer[100], b[2];

//desenha a esteira
j = rand()%5;
for (i = 0; i < 10; i++){
    MoveToEx (hdc, j + 20 + 7*i, 450, NULL);
    LineTo (hdc, j + 24 + 7*i, 450);
    MoveToEx (hdc, j + 20 + 7*i, 500, NULL);
    LineTo (hdc, j + 24 + 7*i, 500);
    j+=5;
    if(j >= 25) j = rand()%5;
}
MoveToEx (hdc, 25, 451, NULL);
LineTo (hdc, 120, 451);
MoveToEx (hdc, 25, 499, NULL);
LineTo (hdc, 120, 499);
Ellipse (hdc, 0, 450, 50, 500);
Ellipse (hdc, 3, 453, 47, 497);
Ellipse (hdc, 95, 450, 145, 500);
Ellipse (hdc, 98, 453, 142, 497);
MoveToEx (hdc, x5 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x5 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, x6 - 25*cos(fi4), y5 - 25*sin(fi4), NULL);
LineTo (hdc, x6 + 25*cos(fi4+pi/2), y5 + 25*sin(fi4+pi/2));
MoveToEx (hdc, 0, 427, NULL);
LineTo (hdc, 23, 450);
if(movest || ge || (xest == 97))
    Rectangle (hdc, xest, 430, xest + 6, 450);
if(npeca || (xnp <= 23))
    Rectangle (hdc, xnp, ynp, xnp + 6, ynp + 20);

//desenha o braço robótico
Rectangle (hdc, 200, 400, 300, 450);
i = 0;
for(cont = 0; cont < 8; cont++){
    MoveToEx(hdc, 200, 450 - cont*i, NULL);
    LineTo(hdc, 300, 450 - cont*i);
    i++;
}
for(i = 0; i < 2; i++){
    MoveToEx (hdc, x3 - (25 - i)*cos(fi3+pi/2), y3 - (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 - (25 - i)*cos(fi3+pi/2), y2 - (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x3 + (25 - i)*cos(fi3+pi/2), y3 + (25 + i)*sin(fi3+pi/2), NULL);
    LineTo (hdc, x2 + (25 - i)*cos(fi3+pi/2), y2 + (25 + i)*sin(fi3+pi/2));
    MoveToEx (hdc, x2 - (25 - i)*cos(fi2+pi/2), y2 - (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 - (25 - i)*cos(fi2+pi/2), y1 - (25 + i)*sin(fi2+pi/2));
    MoveToEx (hdc, x2 + (25 - i)*cos(fi2+pi/2), y2 + (25 + i)*sin(fi2+pi/2), NULL);
    LineTo (hdc, x1 + (25 - i)*cos(fi2+pi/2), y1 + (25 + i)*sin(fi2+pi/2));
}
for(i = 0; i < 6; i++){
    MoveToEx (hdc, x1 + 20 - i, y1, NULL);
    LineTo (hdc, x1 - fr + 20 - i, y1 + 40);
    MoveToEx (hdc, x1 - 20 + i, y1, NULL);
    LineTo (hdc, x1 + fr - 20 + i, y1 + 40);
}
if(brcpeca)
    Rectangle (hdc, x1 - 3, y1 + 30, x1 + 3, y1 + 50);
Ellipse (hdc, 225, 375, 275, 425);
Ellipse (hdc, 228, 378, 272, 422);
MoveToEx (hdc, x3 - 25*cos(fi3), y3 - 25*sin(fi3), NULL);
LineTo (hdc, x3 + 25*cos(fi3), y3 + 25*sin(fi3));
Ellipse (hdc, x2 - 25, y2 - 25, x2 + 25, y2 + 25);
Ellipse (hdc, x2 - 22, y2 - 22, x2 + 22, y2 + 22);
MoveToEx (hdc, x2 - 25*cos (fi2), y2 - 25*sin(fi2), NULL);
LineTo (hdc, x2 + 25*cos(fi2), y2 + 25*sin(fi2));
Ellipse (hdc, x1 - 25, y1 - 25, x1 + 25, y1 + 25);
Ellipse (hdc, x1 - 22, y1 - 22, x1 + 22, y1 + 22);

```

```

//desenha buffer de saída
for(cont = 0; cont < 3; cont++)
    Rectangle (hdc, 350 + cont, 390 + cont, 550 - cont, 450 - cont * 2);
sprintf(buffer, "Número de Peças: %d", numpeças);
TextOut(hdc, 370, 410, buffer, strlen(buffer));

//Apresenta os esquemas
TextOut(hdc, 550, 50, "Esquemas de Cores:", 18);
TextOut(hdc, 600, 68, "1", 1);
TextOut(hdc, 600, 86, "2", 1);
TextOut(hdc, 600, 104, "3", 1);

//Apresenta o texto do lembrete
for(i = 0; i < 5; i++){
    if(i%2 == 0){
        MoveToEx(hdc, i, 20 + i, NULL);
        LineTo(hdc, i, 165 - i);
        LineTo(hdc, 370 - i, 165 - i);
    }
    MoveToEx(hdc, i, 30 + i, NULL);
    LineTo(hdc, 370 - i, 30 + i);
    LineTo(hdc, 370 - i, 165 - i);
}
i = 0;
k = 0;
TextOut(hdc, 100, 37, lembrete, strlen(lembrete));
for(cont = 0; cont < strlen(buff); cont++){
    b[0] = buff[cont];
    b[1] = '\0';
    TextOut(hdc, i*7 + 10, k*16 + 50, b, strlen(b));
    i++;
    if(i == 50){
        i = 0;
        k++;
    }
}
MoveToEx (hdc, 10+xPontolns*7, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7, yPontolns*16 + 64);
MoveToEx (hdc, 10+xPontolns*7+1, 50 + yPontolns*16, NULL);
LineTo(hdc, 10+xPontolns*7+1, yPontolns*16 + 64);
}

void InicializaArq (HWND hjan) {
    static char *szFiltro[] = { "Arquivos TPC (*.TPC)", "*.tpc", "Todos os Arquivos (*.*)", "**.*", "" };

    fname.IStructSize      = sizeof (OPENFILENAME) ;
    fname.hwndOwner        = hjan ;
    fname.hInstance        = NULL ;
    fname.lpstrFilter       = szFiltro [0] ;
    fname.lpstrCustomFilter = NULL ;
    fname.nMaxCustFilter    = 0 ;
    fname.nFilterIndex     = 0 ;
    fname.lpstrFile        = NULL ;
    fname.nMaxFile         = _MAX_PATH ;
    fname.lpstrFileTitle    = NULL ;
    fname.nMaxFileTitle    = _MAX_FNAME + _MAX_EXT ;
    fname.lpstrInitialDir   = NULL ;
    fname.lpstrTitle        = NULL ;
    fname.Flags             = 0 ;
    fname.nFileOffset      = 0 ;
    fname.nFileExtension   = 0 ;
    fname.lpstrDefExt       = "tpc" ;
    fname.lCustData         = 0L ;
    fname.lpfnHook          = NULL ;
    fname.lpTemplateName   = NULL ;
}

void PoeTitulo (HWND hjan, char *filename) {
    char szTit [64 + _MAX_FNAME + _MAX_EXT] ;
    strcpy (szTit, szTitulo);
    strcat (szTit, " - ");
    strcat (szTit, filename [0] ? filename : SEMTITULO);
}

```

```

    SetWindowText (hjan, szTit);
}

BOOL PerguntaSeSalva (HWND hjan, char *filename) {
    char szBuffer [64 + _MAX_FNAME + _MAX_EXT] ;
    BOOL nReturn ;

    szBuffer[0] = '\0';
    strcpy (szBuffer, "Salvar as alterações feitas em ");
    strcat (szBuffer, filename);

    nReturn = MessageBox (hjan, szBuffer, szTitulo, MB_YESNOCANCEL | MB_ICONQUESTION);

    if (nReturn == IDYES)
        if (!SendMessage (hjan, WM_COMMAND, IDM_SALVAR, 0L))
            nReturn = IDCANCEL ;

    return nReturn ;
}

BOOL CALLBACK ProcDlgImpressao (HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam){
switch (msg){
    case WM_COMMAND:
        bUsuarioAborta = TRUE;
        EnableWindow (GetParent (hDlg), TRUE);
        DestroyWindow (hDlg);
        hDlgImpressao = 0;
        return TRUE;
    }
    return FALSE;
}

BOOL CALLBACK ProcAborto (HDC hDCImp, int iCod){
MSG msg;
while (!bUsuarioAborta && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE)){
    if (!hDlgImpressao || !IsDialogMessage (hDlgImpressao, &msg)){
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}
return !bUsuarioAborta;
}

HDC ObtemCDImpress (void) {
    PRINTER_INFO_5 pinfo5[3];
    DWORD dw nec, dw ret;

    if (EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, (LPBYTE) pinfo5, sizeof (pinfo5), &dw nec, &dw ret))
        return CreateDC (NULL, pinfo5[0].pPrinterName, NULL, NULL);

    return 0;
}

BOOL ImprimirArquivo (HINSTANCE hCop, HWND hjan, char *szNomeArq){
static DOCINFO di = {sizeof (DOCINFO), "", NULL};
static PRINTDLG pd;

pd.lStructSize = sizeof (PRINTDLG);
pd.hwndOwner = hjan;
pd.hDevMode = NULL;
pd.hDevNames = NULL;
pd.hDC = NULL;
pd.Flags = PD_ALLPAGES | PD_COLLATE | PD_RETURNDC;
pd.nFromPage = 0;
pd.nToPage = 0;
pd.nMinPage = 0;
pd.nMaxPage = 0;
pd.nCopies = 1;
pd.hInstance = NULL;
pd.lCustData = 0L;
pd.lpfnPrintHook = NULL;
pd.lpfnSetupHook = NULL;
pd.lpPrintTemplateName = NULL;
pd.lpSetupTemplateName = NULL;
pd.hPrintTemplate = NULL;
pd.hSetupTemplate = NULL;

if (!PrintDlg (&pd))

```



```

        return TRUE;
EnableWindow (hjan, FALSE);
bSucesso = TRUE;
bUsuarioAborta = FALSE;
hDlgImpressao = CreateDialog (hCop, (LPCTSTR) "CaixaDlgImpressao", hjan, ProcDlgImpressao);
SetDlgItemText (hDlgImpressao, IDD_NOMEARQ, szNomeArq);
SetAbortProc (pd.hDC, ProcAborto);
GetWindowText (hjan, (PTSTR) di.lpszDocName, sizeof (LPSTR));
if (StartDoc (pd.hDC, &di)>0) {
    if (StartPage (pd.hDC) < 0)
        bSucesso = FALSE;
    display(pd.hDC);
    if (EndPage (pd.hDC) < 0)
        bSucesso = FALSE;
}
else
    bSucesso = FALSE;
if (bSucesso)
    EndDoc (pd.hDC);
if (!bUsuarioAborta){
    EnableWindow (hjan, TRUE);
    DestroyWindow (hDlgImpressao);
}
DeleteDC (pd.hDC);
return !bSucesso && bUsuarioAborta;
}

```

---

#### SEDCURSO.H

---

```

#define IDM_ABRIR          0
#define IDM_SALVAR         10
#define IDM_IMPRIMIR       20
#define IDM_SAIR           30
#define IDM_GE             40
#define IDM_PP             50
#define IDM_GB             60
#define IDM_SB             70
#define IDM_PT             80
#define IDM_E1             90
#define IDM_E2            100
#define IDM_E3            110
#define IDM_MOSTRAR        120
#define IDM_ESCB          130
#define IDM_SOBRE          140
#define IDTB_BMP           300
#define IDM_TOOLBAR        200
#define NUMBUTTONS         7
#define pi                 3.1415
#define ID_TEMPORIZ        1
#define SEMTITULO          "(Sem Título)"
#define IDD_NOMEARQ        7

```

---

#### SEDCURSO.RC

---

```

#include "sedcurso.h"
IDTB_BMP BITMAP "toolbarsedcurso.bmp"
sedcurso MENU {
    POPUP "&Arquivo" {
        MENUITEM "&Abrir\t^A", IDM_ABRIR
        MENUITEM "&Salvar\t^S", IDM_SALVAR
        MENUITEM "&Imprimir\t^N", IDM_IMPRIMIR
        MENUITEM SEPARATOR
        MENUITEM "Sai\r\t^R", IDM_SAIR
    }
}

```

```

POPUP "&Sistema" {
  MENUITEM "&Girar Esteira\t^G", IDM_GE
  POPUP "&Braço Robótico" {
    MENUITEM "&Pegar Peça\t^P", IDM_PP
    MENUITEM "Girar &Braço Robótico\t^B", IDM_GB
    MENUITEM "S&oltar Peça\t^O", IDM_SB
  }
  MENUITEM "P&eças Trabalhadas\t^E", IDM_PT
}
POPUP "&Cores" {
  MENUITEM "Esquema &1\t^H", IDM_E1
  MENUITEM "Esquema &2\t^J", IDM_E2
  MENUITEM "Esquema &3\t^K", IDM_E3
}
POPUP "&Botões" {
  MENUITEM "&Mostrar\t^M", IDM_MOSTRARB
  MENUITEM SEPARATOR
  MENUITEM "Es&conder\t^C", IDM_ESCB
}
POPUP "&Ajuda" {
  MENUITEM "So&bre\t^F1", IDM_SOBRE
}
}

sedcurso ACCELERATORS {
  "^A",          IDM_ABRIR
  "^S",          IDM_SALVAR
  "I",           IDM_IMPRIMIR, CONTROL
  "^R",          IDM_SAIR
  "^G",          IDM_GE
  "^P",          IDM_PP
  "^B",          IDM_GB
  "^O",          IDM_SB
  "^E",          IDM_PT
  "H",           IDM_E1, CONTROL
  "^J",          IDM_E2
  "^K",          IDM_E3
  "^M",          IDM_MOSTRARB
  "^C",          IDM_ESCB
  VK_F1,         IDM_SOBRE, VIRTKEY
}

CaixaDlgImpressao DIALOG 40, 40, 120, 40
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE
{
  CTEXT "Cancelar a Impressão", -1, 4, 6, 120, 12
  DEFPUSHBUTTON "Cancelar", IDCANCEL, 44, 22, 32, 14, WS_GROUP
}

sedcurso ICON "sedcurso.ico"
sedcur1 CURSOR "cur1.cur"
sedcur2 CURSOR "cur2.cur"
sedcur3 CURSOR "cur3.cur"

```

\*  
\*  
\*  
\*  
\*  
\*

Com a introdução dos recursos de impressão, o programa SEDCURSO está completo. Com esta inclusão, a caixa de diálogo comum é apresentada na Figura 10.

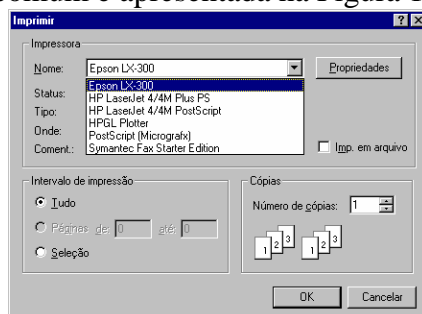


Figura 10

Essa caixa de diálogo comum de impressão é criada quando o usuário seleciona o item **Imprimir** no menu **Arquivo**. Se houver algum problema com a impressão, ou se o usuário desistir de imprimir o trabalho, a caixa de mensagem **Imprime** é apresentada (Figura 11).

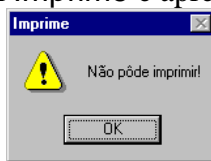


Figura 11

Por outro lado, se o usuário envia o trabalho para a impressão, aparece a caixa de diálogo mostrada na Figura 12, em que o usuário pode pressionar o botão **Cancelar**, caso desista da impressão.

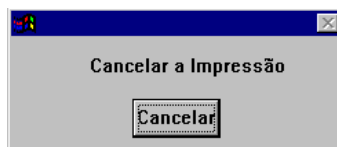


Figura 12

### Código incluído no programa SEDCURSO

Para a impressão, as seguintes declarações de protótipos são incluídas:

```
BOOL CALLBACK ProcAborto (HDC, int);
BOOL ImprimirArquivo (HINSTANCE, HWND, char *);
HDC ObtemCDImpress (void);
```

que são para criar a caixa de diálogo de cancelamento da impressão, para imprimir o arquivo na janela e para obter o identificador do contexto do dispositivo da impressora, respectivamente.

As seguintes variáveis são incluídas para poder ser impressa uma página:

```
BOOL bUsuarioAborta; - variável utilizada para avaliar se o usuário solicitou o
cancelamento da impressão;
HWND hDlgImpressao; - identificador da caixa de diálogo comum de impressão;
BOOL bSucesso; - variável utilizada para definir se houve sucesso na impressão;
```

A função WinMain não é modificada.

O procedimento de janela apresenta as seguintes linhas de código incluídas:

Mensagem WM\_COMMAND:

Quando é solicitado do programa a impressão do conteúdo da janela, é chamada a função de impressão que gera a caixa de diálogo comum de impressão. Se a impressão não é realizada por qualquer motivo, uma caixa de mensagem é apresentada avisando que o programa não conseguiu imprimir:

```
if(ImprimirArquivo (hCop, hjan, filename))
    MessageBox (hjan, "Não pôde imprimir!", szNomeAplic, MB_OK | MB_ICONEXCLAMATION);
```

As funções ligadas à impressão do arquivo são vistas a seguir.

### Caixa de Diálogo "Cancelar"

Quando uma impressão está sendo enviada para a impressora, pode-se especificar um procedimento para que se possa cancelá-la. Assim, o programa deve apresentar uma caixa de diálogo em que o usuário saiba que a impressão está progredindo e que se possa pressionar um botão de cancelamento da impressão caso seja desejado. Esta caixa de diálogo deve ser destruída pelo próprio programa quando a impressão for completamente enviada à impressora.

Essa caixa de diálogo é muito simples, desde que apresenta apenas um controle de comando que é o botão de cancelamento.

Essa função é iniciada com a declaração:

```
BOOL CALLBACK ProcDlgImpressao (HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
```

Essa caixa de diálogo apresenta a seguinte mensagem:

WM\_COMMAND: - se o botão **Cancelar** for pressionado, a variável de controle de cancelamento define que o procedimento de impressão foi cancelado:

```
bUsuarioAborta = TRUE;
```

habilita-se o mouse e o teclado para a janela do programa:

```
EnableWindow (GetParent (hDlg), TRUE);
```

destrói-se a caixa de diálogo:

```
DestroyWindow (hDlg);
```

e elimina-se o seu identificador:

```
hDlgImpressao = 0;
```

### **Cancelamento da Impressão**

Para que a impressão possa ser cancelada, o seguinte procedimento é exigido:

```
BOOL CALLBACK ProcAborto (HDC hDCImp, int iCod)
```

Esse procedimento avalia se o botão da caixa de diálogo **Cancelar** foi pressionado ou se a impressão pode continuar.

Define-se a variável

```
MSG msg;
```

para utilizar nas funções do seguinte laço:

```
while (!bUsuarioAborta && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE)){  
    if (!hDlgImpressao || !IsDialogMessage (hDlgImpressao, &msg)){  
        TranslateMessage (&msg);  
        DispatchMessage (&msg);  
    }  
}
```

Esse laço avalia se a variável de cancelamento da impressão tem valor **TRUE** e se a impressão terminou através da função **PeekMessage**. Observe que a variável **bUsuarioAborta** obtém o valor **TRUE**, se o botão da caixa de diálogo **Cancelar** for pressionado. A função **PeekMessage** avalia as mensagens na fila se **bUsuarioAborta** é **FALSE**. Esta função é semelhante à função **GetMessage**, com a diferença que ela não espera que uma mensagem seja colocada na fila. Seus parâmetros são: 1) ponteiro para a estrutura de mensagem (**MSG**); 2) identificador de janela; 3) valor que especifica a primeira mensagem da fila; 4) valor que identifica a última mensagem da fila e 5) especificação da identificação das mensagens. Se os parâmetros 2, 3 e 4 sendo **NULL** ou zero, indica que as mensagens devem ser retornadas a todas as janelas do programa. O quinto parâmetro pode ter os valores:

- **PM\_REMOVE** – se a mensagem deve ser removida da fila após processá-la;
- **PM\_NOREMOVE** – caso contrário.

Dessa forma, enquanto houver uma mensagem na fila e não for cancelada a impressão, o laço avalia se o identificador da caixa de diálogo **Cancelar** é **TRUE**, ou seja a caixa de diálogo não foi destruída, e se há mensagens direcionadas à esta caixa de diálogo. Se a caixa de diálogo **Cancelar** não foi destruída ou se existem mensagens sendo processada por ela, é mantida uma avaliação dos eventos do teclado e mouse sobre esta caixa de diálogo.

O valor retornado por essa função é

```
return !bUsuarioAborta;
```

que indica se o usuário solicitou ou não, o cancelamento da impressão.

### **Obtenção do Identificador de Impressão**

A função **ObtemCDImpress** obtém o identificador do contexto do dispositivo de impressão.

As variáveis necessárias a esse procedimento são:

**PRINTER\_INFO\_5** pinfo5[3]; - estrutura que especifica as informações detalhadas das impressoras, como nomes das impressoras, nomes das portas de impressão e atributos das impressoras;

**DWORD** dw nec, dw ret; - variáveis utilizadas respectivamente, para: conter o número de bytes copiados ou requeridos para a impressão e para conter o valor do número da estrutura **PRINTER\_INFO\_5**.

A avaliação das impressoras enumeradas através da declaração:

```
if (EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, (LPBYTE) pinfo5, sizeof (pinfo5), &dw nec, &dw ret))
```

retorna um identificador para o contexto da impressora escolhida:

```
return CreateDC (NULL, pinfo5[0].pPrinterName, NULL, NULL);
```

A função **EnumPrinters** enumera todas as impressoras disponíveis. Seus parâmetros são: 1) tipos de impressoras disponíveis; 2) nomes das impressoras (NULL enumera todas as impressoras instaladas); 3) tipo de dado referente à estrutura (nesse caso 5 referenciando-se à estrutura **PRINTER\_INFO\_5**); 4) nome (ponteiro) para a estrutura; 5) tamanho da estrutura apontada pela função **EnumPrinters**; 6) número de bytes copiados para a impressão e 7) ponteiro para o valor que recebe o número da estrutura. A função **CreateDC** cria um identificador de contexto para o dispositivo especificado. Seus parâmetros são: 1) ignorado, devendo ser sempre NULL; 2) nomes dos dispositivos especificados; 3) ignorado, devendo ser sempre NULL e 4) ponteiro para uma estrutura denominada **DEVMODE**, que contém os dados específicos de inicialização do dispositivo para o seu *driver* (o valor NULL deve ser colocado se o *driver* do dispositivo usa o dispositivo padrão).

Caso não haja uma impressora especificada, não é criado um identificador: return 0;

## Função de Impressão

A função de impressão recebe o identificador da cópia do programa, o identificador da janela que está solicitando uma impressão e o nome do arquivo a ser impresso:

BOOL ImprimirArquivo (HINSTANCE hCop, HWND hjan, char \*szNomeArq)

As variáveis utilizadas são:

**static DOCINFO di = {sizeof (DOCINFO), "", NULL};** - estrutura que contém os arquivos de entrada e saída e outras informações usadas pela função **StartDoc** (discutida adiante). Essa estrutura é declarada com os dados: 1) tamanho da estrutura; 2) nome do documento e 3) nome do arquivo de saída.

**static PRINTDLG pd;** - estrutura que inicializa a caixa de diálogo comum de impressão;

A caixa de diálogo comum de impressão é inicializada nas declarações da estrutura **PRINTDLG**, como a seguir:

```
pd.lStructSize      = sizeof (PRINTDLG);
pd.hwndOwner        = hjan;
pd.hDevMode         = NULL;
pd.hDevNames        = NULL;
pd.hDC              = NULL;
pd.Flags            = PD_ALLPAGES | PD_COLLATE | PD_RETURNDC;
pd.nFromPage        = 0;
pd.nToPage          = 0;
pd.nMinPage         = 0;
pd.nMaxPage         = 0;
pd.nCopies          = 1;
pd.hInstance        = NULL;
pd.lCustData        = 0L;
pd.lpfnPrintHook     = NULL;
pd.lpfnSetupHook    = NULL;
pd.lpPrintTemplateName = NULL;
pd.lpSetupTemplateName = NULL;
pd.hPrintTemplate   = NULL;
pd.hSetupTemplate   = NULL;
```

Os principais parâmetros a serem considerados para descrição nessa estrutura são respectivamente: 1) tamanho da estrutura; 2) identificador da janela que chamou a caixa de diálogo comum de impressão; 3) só utilizado se utilizar a estrutura **DEVMODE** para inicializar a caixa de diálogo de impressão; 4) observação igual ao terceiro parâmetro; 5) identifica o contexto do dispositivo (este valor é obtido na função **ObtemCDImpress**); 6) valores de inicialização da caixa de diálogo comum de impressão, em que são definidos quais itens são habilitados (nesse caso, caixa de seleção para todas as páginas, cópias agrupadas e admite a mudança do valor de **hDC** através da função **ObtemCDImpress**, respectivamente); 7) valor inicial da borda de impressão na página; 8) valor final da borda de impressão na página; 9) valor mínimo para a faixa de impressão definidas nos itens 7 e 8; 10) valor máximo para a faixa de impressão definidas nos itens 7 e 8; 11) número de cópias a serem impressas; 12) identificador da cópia do programa que chamou a caixa de diálogo comum de impressão. Os demais valores podem ser sempre colocados como NULL, ou 0L no caso de **lCustData**.

Se o botão **Cancelar** é pressionado na caixa de diálogo comum de impressão, a impressão é cancelada, sendo apresentada a caixa de mensagem com o texto **Não pôde imprimir o arquivo!** (ver mensagem **WM\_COMMAND**):

```
if (!PrintDlg (&pd))  
    return TRUE;
```

Desabilita-se o teclado e o mouse para a janela principal:

```
EnableWindow (hjan, FALSE);
```

e inicializam-se as variáveis de controle:

```
bSucesso = TRUE;  
bUsuarioAborta = FALSE;
```

Obtém-se o identificador da caixa de diálogo **Cancelar**, chamando-a:

```
hDlgImpressao = CreateDialog (hCop, (LPCTSTR) "CaixaDlgImpressao", hjan, ProcDlgImpressao);
```

Coloca-se o nome do arquivo na caixa de diálogo **Cancelar**:

```
SetDlgItemText (hDlgImpressao, IDD_NOMEARQ, szNomeArq);
```

e inicializa o procedimento de cancelamento:

```
SetAbortProc (pd.hDC, ProcAborto);
```

Copia-se o texto da barra de título para a caixa de diálogo **Cancelar**:

```
GetWindowText (hjan, (PTSTR) di.lpszDocName, sizeof (LPSTR));
```

e inicia-se a impressão:

```
if (StartDoc (pd.hDC, &di)>0)
```

A função **StartDoc** inicia o trabalho de impressão. Se a impressora não estiver preparada para receber os dados, a impressão é cancelada:

```
if (StartPage (pd.hDC) < 0)  
    bSucesso = FALSE;
```

Caso contrário é iniciada a impressão:

```
display(pd.hDC);
```

Se a página foi finalizada, e não há mais páginas para imprimir, retorna o controle ao programa:

```
if (EndPage (pd.hDC) < 0)  
    bSucesso = FALSE;
```

A função **EndPage** informa ao dispositivo que o programa está finalizando a impressão de uma página.

Se a impressão continua até finalizar:

```
if (bSucesso)  
    EndDoc (pd.hDC);
```

e caso haja erro,

```
else  
    bSucesso = FALSE;
```

a impressão pára.

Em qualquer momento que for cancelada a impressão, o controle do mouse e do teclado é devolvido ao programa e a caixa de diálogo é destruída:

```
if (!bUsuarioAborta){  
    EnableWindow (hjan, TRUE);  
    DestroyWindow (hDlgImpressao);  
}
```

Por fim, é eliminado o contexto do dispositivo de impressão e é retornado o valor que determina se o programa apresenta a caixa de mensagem com o texto **Não Pôde imprimir!:**

```
DeleteDC (pd.hDC);  
return !bSucesso && bUsuarioAborta;
```

## Arquivo **SEDCURSO.H** e **SEDCURSO.RC**

O arquivo **SEDCURSO.H** inclui apenas a definição do identificador do arquivo para impressão: **IDD\_NOMEARQ**. O arquivo **SEDCURSO.RC** inclui o código referente à construção da caixa de diálogo: **Cancelar a Impressão**.

Uma caixa de diálogo é construída como a seguir:

Em primeiro lugar, deve-se definir:

1. Nome da caixa de diálogo igual ao segundo parâmetro da função **CreateDialog**, ou da função **DialogBox** (no arquivo **.C**) seguido do tipo de recurso definido pela macro **DIALOG**

e dos valores  $x$ ,  $y$  onde a caixa de diálogo deve aparecer em relação à janela principal, além de sua largura e altura;

2. Estilo da caixa de diálogo: macro **STYLE** seguido dos estilos definidos para a janela da caixa de diálogo;
3. Título da caixa de diálogo: macro **CAPTION** seguido do título entre aspas duplas.
4. Fonte da caixa de diálogo: macro **FONT** seguido do nome da fonte entre aspas duplas;

Por exemplo:

```
CordoFundo DIALOG 8, 17, 124, 167
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE
CAPTION "Escolher a Cor da Janela"
FONT 8, "MS Sans Serif"
```

cria uma caixa de diálogo **Escolher a Cor da Janela**, que é chamada quando a função **CreateDialog** contém o segundo parâmetro dado por **CordoFundo**. Esta caixa de diálogo é apresentada na posição  $x=8$ ,  $y=17$ , e tem largura 124 e altura 167. seu estilo é uma janela com título, menu de sistema e visível. Todas as palavras são apresentadas com a fonte **MS Sans Serif**, tamanho 8.

Cada controle criado em uma caixa de diálogo é gerado através da macro **CONTROL**, seguido de seu título ou legenda entre aspas duplas, seu identificador (se for -1, deve ser um controle estático, pois não envia mensagem **WM\_COMMAND**), tipo de controle (static, scrollbar, radiobutton, button, etc.), bits de estilo do controle (específico para cada um: **SS\_** controle estático, **SBS\_** barra de rolagem, **BS\_** botão, **ES\_** caixa de edição, etc.), e respectivas posições iniciais ( $x$  e  $y$ ) e largura e altura. Por exemplo:

```
CONTROL "&Vermelho", -1, "static", SS_CENTER, 10, 4, 30, 8
```

é um controle estático que apresenta apenas um texto centralizado;

```
CONTROL "", 10, "scrollbar", SBS_VERT | WS_TABSTOP, 16, 17, 20, 100
```

é uma barra de rolagem vertical com identificador 10;

```
CONTROL "Ok", IDOK, "button", BS_DEFPUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 20, 34, 48, 14
```

é um botão de pressionamento com título **OK**, identificador **IDOK**, definido como pronto para pressionar;

```
CONTROL "&Quadrado", IDC_QUADRADO, "button", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 8, 43, 45, 12
```

é um radiobutton com identificador **IDC\_QUADRADO** e título **Quadrado**.

Todos os números no final de cada declaração indicam a posição inicial e as dimensões do controle.

Outros tipos de controle podem ser utilizados, como: **CTEXT**, **ICON**, **DEFPUSHBUTTON**, **PUSHBUTTON**, etc, como utilizados na criação da caixa de diálogo **Escolher uma Figura**. Suas definições são as mesmas utilizadas com a macro **CONTROL**, mudando apenas a ordem: tipo, título, identificador, posições iniciais ( $x$  e  $y$ ), altura e largura e bits de estilo. Por exemplo:

```
CTEXT "Dialogo", -1, 64, 13, 64, 13, SS_CENTER | SS_SUNKEN | WS_BORDER | WS_GROUP
```

indica uma caixa de texto estática (o texto é **Diálogo** e não pode ser mudado—identificador-1), na posição  $x=64$ ,  $y=13$ , com largura igual a 64 e altura igual a 13, com seu respectivo estilo: texto centralizado, afundado na janela, com borda e pertencente a um grupo;

```
ICON "Dialogo", -1, 8, 4, 23, 21, SS_ICON | SS_SUNKEN
```

é um ícone que apresenta seu nome dado como **Dialogo** (observe que este é o nome do ícone do programa na última linha do código **DIALOGO.RC**), estático (não muda), na sua devida posição e com as definições do estilo: é um ícone (pode ser um bitmap) e afundado na janela;

```
GROUPBOX "&Figura", -1, 4, 30, 54, 112
```

borda de um grupo de controles com título fixo **Figura** (identificador -1), posições e dimensões;

```
PUSHBUTTON "Cancelar", IDC_CANCEL, 80, 151, 40, 14
```

botão com título **Cancelar**, com identificador **IDC\_CANCEL**, posições e dimensões;

```
DEFPUSHBUTTON "OK", IDC_OK, 20, 151, 40, 14
```

botão definido como selecionado, onde o pressionamento da tecla **ENTER** já define o envio de uma mensagem para ele, com título **OK**, posições iniciais e dimensões.

Observe o leitor que para cada caixa de diálogo, deve ser criado seu procedimento de janela no arquivo principal (.C), contendo o código para cada controle criado nela. Maiores informações sobre a construção de caixas de diálogo são encontradas na bibliografia sugerida ao final desta apostila.

Com essas informações, todas as caixas de diálogo criadas para programas WINDOWS são definidas. A construção de caixas de diálogo segue estes padrões para qualquer tipo de controle que seja necessário utilizar. Assim, o código da caixa de diálogo Cancelar a Impressão é facilmente entendida, pois ela é uma das mais simples que há, como o próprio leitor pode analisar.

```
CaixaDlgImpressao DIALOG 40, 40, 120, 40
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE
{
    CTEXT "Cancelar a Impressão", -1, 4, 6, 120, 12
    DEFPUSHBUTTON "Cancelar", IDCANCEL, 44, 22, 32, 14, WS_GROUP
}
```



## BIBLIOGRAFIA

1. Charles Petzold, Programando para Windows 95, Tradução: *Jeremias René Descarts Pereira dos Santos*, MAKRON Books do Brasil Editora Ltda., São Paulo - SP, 1996.
2. Herbert Schildt, Programando em C e C++ com Windows 95, Tradução: *Lars Gustav Erick Unonius*, MAKRON Books do Brasil Editora Ltda., São Paulo - SP, 1996.
3. Herbert Schildt, Programando em Windows 95, Segredos e Soluções para Programadores Experientes, Tradução: *Ariosvaldo Griesi*, MAKRON Books do Brasil Editora Ltda., São Paulo - SP, 1997.
4. Borland C++ Help.
5. Eduard Montgomery Meira Costa, Apostila: Programação em C para WINDOWS, 2002.