B-Erase will be nonzero if the background of the window needs to be erased. However, as long as you specified a background brush when you created the window, you can ignore the fErase member. Windows NT will erase the window for you.

2-nWinMode determines how the window will be displayed when the program begins execution. Subsequent calls can display (or remove) the window as necessary. Some common values for nHow are shown here:

<table>
<thead>
<tr>
<th>Display Macros</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_HIDE</td>
<td>Removes the window</td>
</tr>
<tr>
<td>SW_MINIMIZE</td>
<td>Minimizes the window into an icon</td>
</tr>
<tr>
<td>SW_MAXIMIZE</td>
<td>Maximizes the window</td>
</tr>
<tr>
<td>SW_RESTORE</td>
<td>Returns a Window to normal size</td>
</tr>
</tbody>
</table>

3- HWND is a 32-bit integer that is used as a window handle that used contain all feature of window.

4- To set the double-click interval, use SetDoubleClickTime( ). Its prototype is shown here:

   BOOL SetDoubleClickTime ( UINT interval );

5- WM_COMMAND

WM_COMMAND message is sent and the value in LOWORD(wParam) contains the ID of the menu item selected or the accelerator key pressed.
However, using only the value in `LOWORD(wParam)` it is not possible to determine which event occurred. In most situations, it doesn't matter whether the user actually made a menu selection or just pressed an accelerator keystroke a `WM_COMMAND` message is sent to `DialogFunc()`, and `LOWORD(wParam)` contains the ID

Q2:

. (This message will also be sent when your window is first displayed.) Each time your program receives this message it must redisplay the contents of the window.

Before explaining how to respond to a `WM_PAINT` message it might be useful to explain why Windows does not automatically rewrite your window. The answer is short and to the point:. In many situations, it is easier for your program, which has intimate knowledge of the contents of the window, to rewrite it than it would be for Windows to do so. While the merits, of this approach have been much debated by programmers, you should simply accept it, because it is unlikely to change.

The first step to processing a `WM_PAINT` message is to add it to the `switch` statement inside the window function. For example, here is one way to add a `WM_PAINT` case to the previous program

```c
case WM_PAINT: /* process a repaint request */
    hdc = BeginPaint(hwnd, &paintstruct); /* get DC */
    TextOut(hdc, 0, 0, str, strlen(str));
    EndPaint(hwnd, &paintstruct); /* release DC */
    break;
```

Let's look at this closely. First, notice that a device context is obtained using a call to `BeginPaint()` instead of `GetDC()`. For various reasons, when you process a `WM_PAINT` message, you must obtain a device context using `BeginPaint()`, which has this prototype:

```
HDC BeginPaint(HWND hwnd, PAINTSTRUCT *lpPS);
```
BeginPaint( ) returns a device context if successful or NULL on failure. Here, hwnd is the handle of the window for which the device context is being obtained. The second parameter is a pointer to a structure of type PAINTSTRUCT. On return, the structure pointed to by lpPS will contain information that your program can use to repaint the window. PAINTSTRUCT is defined like this:

```
typedef struct tagPAINTSTRUCT {
    HDC hdc; /* handle to device context */
    BOOL fErase; /* true if background must be erased */
    RECT rcPaint; /* coordinates of region to redraw */
    BOOL fRestore; /* reserved */
    BOOL fIncUpdate; /* reserved */
    BYTE rgbReserved[32]; /* reserved */
} PAINTSTRUCT;
```

Here, hdc will contain the device context of the window that needs to be repainted. This DC is also returned by the call to BeginPaint( ). fErase will be nonzero if the background of the window needs to be erased. However, as long as you specified a background brush when you created the window, you can ignore the fErase member. Windows NT will erase the window for you.

The type RECT is a structure that specifies the upper left and lower right coordinates of a rectangular region. This structure is shown here:

```
typedef tagRECT {
    LONG left, top; /* upper left */
    LONG right, bottom; /* lower right */
} RECT;
```

In PAINTSTRUCT, the rcPaint element contains the coordinates of the region of the window that needs to be repainted. For now, you will not need to use the contents of rcPaint because you can assume that the entire window must be repainted. However, real programs that you write will probably need to utilize this information.
Once the device context has been obtained, output can be written to the window. After the window has been repainted, you must release the device context using a call to `EndPaint()`, which has this prototype:

```
BOOL EndPaint(HWND hwnd, CONST PAINTSTRUCT *lpPS);
```

`EndPaint()` returns nonzero. (It cannot fail.) Here, `hwnd` is the handle of the window that was repainted. The second parameter is a pointer to the `PAINTSTRUCT` structure used in the call to `BeginPaint()`.

It is critical to understand that a device context obtained using `BeginPaint()` must be released only through a call to `EndPaint()`. Further, `BeginPaint()` must only be used when a `WM_PAINT` message is being processed. Here is the full program that now processes `WM_PAINT` messages.

### Q3

As explained earlier, Windows NT communicates with your program by sending it messages. All Windows NT applications must establish a *message loop* inside the `WinMain()` function. This loop reads any pending message from the application's message queue and then dispatches that message back to Windows NT, which then calls your program's window function with that message as a parameter. This may seem to be an overly complex way of passing messages, but it is, nevertheless, the way all Windows programs must function. (Part of the reason for this is to return control to Windows NT so that the scheduler can allocate CPU time as it sees fit rather than waiting for your application's time slice to end.)

```c
/* create the message loop. */

while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg); /* allow use of keyboard */
    DispatchMessage(&msg); /* return control to window NT */

    return msg.wParam; /* end of WinMain() */
}
```

```
/* after update */

while (GetMessage(&msg, NULL, 0, 0))
{
    if (!IsDialogMessage(hDlg, &msg))
```
Q4 a-

/* Define a window class. */

wcl.cbSize = sizeof(WNDCLASSEX);

wcl.hInstance = hThisInst; /* handle to this instance */

wcl.lpszClassName = szWinName; /* window class name */

wcl.lpfnWndProc = WindowFunc; /* window function */

wcl.style = 0; /* default style */

wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION); /* standard icon */

wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO); /* small icon */

wcl.hCursor = LoadCursor(NULL, IDC_ARROW); /* cursor style */

wcl.lpszMenuName = NULL; /* no menu */

wcl.cbClsExtra = 0; /* no extra */

wcl.cbWndExtra = 0; /* information needed */

/* Make the window background white. */

wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);

/* Register the window class. */

if (!RegisterClassEx(&wcl)) return 0;

/* Now that a window class has been registered, a window can be created. */
hwnd = CreateWindow(

szWinName, /* name of window class */

"Windows NT Skeleton", /* title */

WS_OVERLAPPEDWINDOW, /* window style - normal */

CW_USEDEFAULT, /* X coordinate - let Windows decide */

CW_USEDEFAULT, /* Y coordinate - let Windows decide */

CW_USEDEFAULT, /* width - let Windows decide */

CW_USEDEFAULT, /* height - let Windows decide */

HWND_DESKTOP, /* no parent window */

NULL,

hThisInst, /* handle of this instance of the program */

NULL /* no additional arguments */);

WNDCLASSEX structure. Its fields are shown here.

UINT cbSize; /* size of the WNDCLASSEX structure */

UINT style; /* type of window */

WNDPROC pfnWndProc; /* address to window func */

int cbClsExtra; /* extra class info */

int cbhhWndlExtra; /* extra window info */

HINSTANCE hInstance; /* handle of this instance */

HICON hIcon; /* handle of standard icon */

HICON hIconSm; /* handle of small icon */

HICON hCursor; /* handle of mouse cursor */

HBRUSH hbrBackground; /* background color */

LPCSTR lpszMenuName; /* name of main menu */

LPCSTR lpszClassName; /* name of window class */
Q4 b

SendDlgItemMessage( ) contains the information requested by IDMsg.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB_ADDSTRING</td>
<td>Adds a string (selection) to the list box.</td>
</tr>
<tr>
<td>LB_GETCURSEL</td>
<td>Requests the index of the selected item.</td>
</tr>
<tr>
<td>LB_SETCURSEL</td>
<td>Selects an item.</td>
</tr>
<tr>
<td>LBJ-FINDSTRING</td>
<td>Finds a matching entry.</td>
</tr>
<tr>
<td>LB_SELECTSTRING</td>
<td>Finds a matching entry and selects it.</td>
</tr>
<tr>
<td>LB_GETTEXT</td>
<td>Obtains the text associated with an item</td>
</tr>
</tbody>
</table>

Q5:

MyDB DIALOG 10, 10, 210, 110

CAPTION "Books Dialog Box"

STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU

{DEFPUSHBUTTON "Author", IDD_AUTHOR, 11, 10, 36, 14
  WS_CHILD | WS_VISIBLE | WS_TABSTOP
PUSHBUTTON "Publisher", IDD_PUBLISHER, 11, 34, 36, 14
  WS_CHILD | WS_VISIBLE | WS_TABSTOP
PUSHBUTTON "Copyright", IDD_COPYRIGHT, 11, 58, 36, 14
  WS_CHILD | WS_VISIBLE | WS_TABSTOP
PUSHBUTTON "Cancel", IDCANCEL, 11, 82, 36, 16,
  WS_CHILD | WS_VISIBLE | WS_TABSTOP
LISTBOX IDl LB1, 60, 5, 140, 33, LBS_NOTIFY | WS_VISIBLE|
  WS_BORDER | WS_VSCROLL | WS_TABSTOP
PUSHBUTTON "Select Book", IDD_SELECT, 103, 41, 54, 14,
case IDD_DONE: /* Title Search button pressed */

/* get current contents of edit box */ GetDlgItemText (hwnd, IDD_EB1, str, 80);

/* find a matching string in the list box */
i = SendDlgItemMessage( hwnd, IDD_LB1, LB_FINDSTRING, 0, (LPARAM) str) ;

if( i != LB_ERR) { /* if match is found */
    /* select the matching title in list box */
    SendMessage(hwnd, IDD_LB1, LB_SETCURSEL, i, 0);
    /* get string associated with that index */
    SendMessage(hwnd, IDD_LB1, LB_GETTEXT, i, (LPARAM) str);
    /* update text in edit box */
    SetDlgItemText (hwnd, IDD_EB1, str) ;
} else MessageBox (hwnd, str, "No Title Matching", MB_OK) ; return 1 ;

switch(message) { case WM_COMMAND:

    switch(LOWORD(wParam)){

    case ID_CANCEL: EndDialog(hwnd, 0); return 1;

    case IDD_COPYRIGHT:

        i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
        sprintf(str, "%u", books[i].copyright);
        MessageBox(hwnd, str, "Copyright", MB_OK); return 1;

    case IDD_AUTHOR:

        i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
        sprintf(str, "%s", books[i].author);
        MessageBox(hwnd, str, "Author", MB_OK); return 1;
case IDD_PUBLISHER:
    i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
    sprintf(str, "%s", books[i].publisher);
    MessageBox(hwnd, str, "Publisher", MB_OK); return 1;

    case IDD_DONE: /* Title Search button pressed */
    /* get current contents of edit box */ GetDlgItemText(hwnd, IDD_EB1, str, 80);
    /* find a matching string in the list box */
    i = SendDlgItemMessage(hwnd, IDD_LB1, LB_FINDSTRING, 0, (LPARAM) str);
    if(i != LB_ERR) { /* if match is found */
        /* select the matching title in list box */
        SendDlgItemMessage(hwnd, IDD_LB1, LB_SETCURSEL, i, 0)
        /* get string associated with that index */
        SendDlgItemMessage(hwnd, IDD_LB1, LB_GETTEXT, i, (LPARAM) str);
        /* update edit box */
        SetDlgItemText(hwnd, IDD_EB1, str);
    }
    else MessageBox(hwnd, str, "No Title Matching", MB_OK); return 1;

    case IDD_LB1: /* process a list box LBN_DBLCLK */
    /* see if user made a selection */
    if(HIWORD(wParam)==LBN_DBLCLK) {
        i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
        sprintf(str, "%s
%s
%s, %u", books[i].title, books[i].author, books[i].publisher, books[i].copyright);
        MessageBox(hwnd, str, "Selection Mode", MB_OK);
        /* get string associated with that index */
        SendDlgItemMessage(hwnd, IDD_LB1, LB_GETTEXT, i, (LPARAM) str);
        /* update edit box */
        SetDlgItemText(hwnd, IDD_EB1, str);
    } return 1;

    case IDD_SELECT: /* Select book button has been pressed */
    i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
    sprintf(str, "%s
%s
%s
%s, %u", books[i].title, books[i].author, books[i].publisher, books[i].copyright); MessageBox(hwnd, str, "Selection Mode", MB_OK);
    /* get string associated with that index */
    SendDlgItemMessage(hwnd, IDD_LB1, LB_GETTEXT, i, (LPARAM) str);
    /* update edit box */
    SetDlgItemText(hwnd, IDD_EB1, str); } return 1;

    case IDD_SELECT: /* Select book button has been pressed */
    i = SendDlgItemMessage(hwnd, IDD_LB1, LB_GETCURSEL, 0, 0); /* get index */
    sprintf(str, "%s
%s
%s
%s, %u", books[i].title, books[i].author, books[i].publisher, books[i].copyright); MessageBox(hwnd, str, "Selection Mode", MB_OK);
    /* get string associated with that index */
case WM_INITDIALOG: /* initialize list box */
for (i = 0; i < NUMBOOKS; i++)
    SendDlgItemMessage(hdwnd, IDD_LB1, LB_ADDSTRING, 0, (LPARAM)books[i].title);
    /* select first item */ SendDlgItemMessage(hdwnd, IDD_LB1, LB_SETCURSEL, 0, 0);
    /* initialize the edit box */ SetDlgItemText(hdwnd, IDD_EB1, books[0].title);
return 1;
return 0;

Q6

switch(message)
{
  case WM_COMMAND:
    switch(LOWORD(wParam))
    {
    case IDM_OPEN: MessageBox(hwnd, "Open File", "Open", MB_OK); break;
    case IDM_CLOSE: MessageBox(hwnd, "Close File", "Close", MB_OK); break;
    case IDM_EXIT:
      response = MessageBox(hwnd, "Quit the Program?", "Exit", MB_YESNO);
      if(response == IDYES) PostQuitMessage(0); break;
    case IDM_COLORS: MessageBox(hwnd, "Set Colors", "Colors", MB_OK); break;
    case IDM_LOW: MessageBox(hwnd, "Low", "Priority", MB_OK); break;
    case IDM_HIGH: MessageBox(hwnd, "High", "Priority", MB_OK); break;
    case IDM_RESOLUTION: MessageBox(hwnd, "Resolution Options", "Resolution", MB_OK); break;
    case IDM_FONT: MessageBox(hwnd, "Font Options", "Fonts", MB_OK); break;
    case IDM__TIME: /* show time */ t = time(NULL);
      tod = localtime(&t); strcpy(str, asctime(tod));
      str[strlen(str)-1] = '\0'; /* remove /\n */

MessageBox(hwnd, str, "Time and Date", MB_OK); break;

case IDM_HELP: MessageBox(hwnd, "No Help", "Help", MB_OK); break;

default: return DefWindowProc(hwnd, message, wParam, IParam); return 0; }

#define IDM_file 100
#define IDM_help 101
#define IDM_LOW 102
#define IDM_HIGH 102
#define IDM_COLORS 103
#define IDM_RESOLUTION 107
#define IDM_HELP 108

MyMenu MENU
{
MENUITEM "&FILE" t F2", IDM_FILE

POPUP "&OPTION" {MENUITEM "&COLOR" t Ctrl-C", IDM_COLORS
POPUP "&Priority" {MENUITEM "&Low" t F4", IDM_LOW

MENUITEM "&High" t F5", IDM_HIGH}

MENUITEM "&Font" t Ctrl-F", IDM_FONT

MENUITEM "&Resolution" t Ctrl-R", IDM_RESOLUTION }

MENUITEM "&FONT", IDM_FONT

MENUITEM "&RESOLUTION", IDM_RESOLUTION }

MENUITEM "&Help", IDM_HELP}