

```

/*****
/*
/*----- P Y T H A G O . C -----*/
/* Task      : Shows swapping of functionality in a thread.
/*             In this example, a thread is used to draw the tree
/*             of Pythagoras.
/*-----*/
/* Authors      : Michael Tischer and Bruno Jennrich
/* developed on   : 06/06/1995
/* last update    : 08/21/1995
/*****
#include <windows.h>
#include <math.h>

#include "turtle32.h"
#include "resource.h"

/*= Constants and Typedefs =====*/

#define MIN_EDGELENGTH 4

/* Parameter structure for tree of Pythagoras -----*/
typedef struct tagPYTHOPARMS
{
    HWND      hWnd;      /* Window in which the PainterThread is to paint */
    float      fAlpha;    /* Angle of "pythagorean" triangle */
    COLORREF   crColor;   /* color to be used */
    long       lSleep;    /* Pause in ms after completing a tree */
} PYTHOPARMS;
typedef PYTHOPARMS *PPYTHOPARMS;

/*= Global variables =====*/

BOOL bTerminateThread; /* True, if thread is to terminate */
BOOL bStartAgain;      /* True, if tree is to be redrawn */
HANDLE hThread;         /* Handle of thread */

/* Global Pythagoras parameters -----*/
PYTHOPARMS PP;

/*****
/* Pythagoras: Draw subtree of tree of Pythagoras
/*-----*/
/* Parameter:      pTC      : TurtleContext
/*                  lEdge   : Length of edge of subtree to be drawn
/*                  pPP      : Parameters (angle, etc. )
/*                  bDraw    : Draw subtree or calculate only coordinates
/*                           for BoundingRect (see Turtle32)
/* Return value: none
/*****
void Pythagoras( PTURTLECONTEXT pTC,
                LONG Edge,
                BOOL bDraw )
{
    if( Edge < MIN_EDGELENGTH ) return; /* Subtree too small */
    /* Exit recursion for redraw or end of thread -----*/
    if( bTerminateThread || bStartAgain ) return;

    turtleForward( pTC, ( float )Edge, bDraw ); /* lower square edge */
    turtleRotate( pTC, ( float )90.0 );

    turtleForward( pTC, ( float )Edge, bDraw ); /* right edge */
    turtleRotate( pTC, ( float )90.0 );

    turtleForward( pTC, ( float )Edge, bDraw ); /* upper edge */

    turtlePush( pTC ); /*the triangle is added here, so save turtle */

    turtleRotate( pTC, ( float )90.0 );
    turtleForward( pTC, ( float )Edge, bDraw ); /* left edge */

    turtlePop( pTC ); /* back to end of upper edge */

    /* Set orientation of turtle according to angle of triangle --*/
    turtleRotate( pTC, -( 180 - PP.fAlpha ) );

```

```

/* save current status, because now the left subtree -----*/
/* is going to be drawn. -----*/
turtlePush( pTC );

/* Recursion via left subtree -----*/
Pythagoras( pTC, ( long )(cos( ( PP.fAlpha * PI ) / 180.0 ) * Edge),
            bDraw );
turtlePop( pTC );

/* Go to apex of right angle -----*/
turtleForward( pTC,
              ( float )(cos( ( PP.fAlpha * PI ) / 180.0 ) * Edge),
              FALSE );
turtleRotate( pTC, ( float )-90.0 );

/* Recursion via right subtree -----*/
Pythagoras( pTC, ( long )(sin( ( PP.fAlpha * PI ) / 180.0 ) * Edge),
            bDraw );
}

/*****
/* threadPainterFunction: Workhorse of PainterThread */
/*-----*/
/* Parameter: lpStart : not required */
/* Return value: Thread Exit Code */
/*-----*/
/* Info: This thread is created after the parameters are entered in */
/* a dialog box and continuously paints the tree of Pythagoras. */
/*****
DWORD WINAPI threadPainterFunction( LPVOID lpStart )
{
    TURTLECONTEXT TC;

    turtleInit( &TC ); /* Initialize TurtleContext */

    SetThreadPriority( GetCurrentThread(), THREAD_PRIORITY_BELOW_NORMAL );

    /* Initialize the turtle of this thread -----*/
    turtleSetWindow( &TC, PP.hWnd );

    /* Set character color or character pen -----*/
    turtleSetPen( &TC, PP.crColor, 1 );

    /* First run a "blind" pass to determine the */
    /* drawing area. */
    turtleInitBounding( &TC );

    turtleMoveTo( &TC, 0, 0, FALSE ); /* Set starting position */
    turtleSetAngle( &TC, ( float )0.0 );
    /* Draw the tree once and determine the bounding rectangle */
    Pythagoras( &TC, 100, FALSE );

    /* Terminate the thread here already? -----*/
    turtleUseBounding( &TC, TRUE );

    /* End thread if application requires it -----*/
    while( !bTerminateThread )
    {
        InvalidateRect( PP.hWnd, NULL, TRUE ); /* Clear background */
        UpdateWindow( PP.hWnd );

        turtleMoveTo( &TC, 0, 0, FALSE ); /* Draw new tree */
        turtleSetAngle( &TC, ( float )0.0 );
        Pythagoras( &TC, 100, TRUE );
        bStartAgain = FALSE;
    }
    turtleExit( &TC ); /* Release TurtleContext */
    return 0;
}

/*****
/* DialogProc : Window function of the dialog box */
/*-----*/
/* Parameter: default dialog box parameters */
/* Return value: default dialog box return values */
/*****/

```

[illegible]

```

        CHOOSECOLOR cc;
        COLORREF crCustom[ 16 ];
        cc.lStructSize = sizeof( CHOOSECOLOR );
        cc.hwndOwner = hWnd;
        cc.hInstance = NULL;
        cc.rgbResult = PP.crColor;
        cc.lpCustColors = crCustom;
        cc.Flags = CC_PREVENTFULLOPEN | CC_RGBINIT;
        cc.lCustData = 0L;
        cc.lpfHook = NULL;
        cc.lpTemplateName = NULL;

        if( ChooseColor( &cc ) )                /* Color chosen? */
        {
            PP.crColor = cc.rgbResult;            /* Note color... */
            InvalidateRect( hWnd, NULL, FALSE ); /* ..and display */
        }
        break;
    }
    break;
}
return FALSE;
}

/*****
 * WndProc : Window function of the main window
 */
-----
/* Parameter:      Default window parameters
 * Return value:   Default window return value
 */
-----
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_CREATE:
            hThread = NULL;                /* still no thread */
            PP.fAlpha = ( float )30;      /* Default values */
            PP.crColor = RGB(0,0,0);
            break;

        case WM_COMMAND:                  /* Which menu item was selected? */
            switch( LOWORD( wParam ) )
            {
                case IDM_EXIT:             /* Good-Bye! */
                    DestroyWindow( hWnd );
                    break;
                case IDM_NEW:              /* New parameters for thread */
                    if( DialogBox( NULL,
                                   MAKEINTRESOURCE( IDD_PARAMETER ),
                                   hWnd,
                                   DialogProc ) )
                    { /* Exit dialog box with OK ? -----*/
                        DWORD dwThreadId;

                        if( hThread ) /* end any existing thread -----*/
                        {
                            bTerminateThread = TRUE;                /* Set flag... */
                            /* ...and wait for end -----*/
                            WaitForSingleObject( hThread, INFINITE );
                        }

                        hThread = NULL;    /* Mark thread handle as invalid */
                        PP.hWnd = hWnd;    /* Output window in global structure */

                        InvalidateRect( hWnd, NULL, TRUE );
                        UpdateWindow( hWnd );                /* Clear window contents */

                        /* do not terminate thread immediately... -----*/
                        bTerminateThread = FALSE;

                        /* ..and the tree will be repainted anyway -----*/
                        bStartAgain = FALSE;

                        /* Create thread -----*/
                        hThread = CreateThread( NULL,

```

```

        OL,
        threadPainterFunction,
        NULL,
        OL,
        &dwThreadID );
    if( hThread == NULL )
        MessageBox( hWnd,
            "Could not create thread!",
            "Error",
            MB_OK );
    }
    break;
}
break;

case WM_SIZE:
    bStartAgain = TRUE;                /* Draw tree from beginning */
    break;

case WM_DESTROY: /* Close window and terminate thread -----*/
    if( hThread )
    {
        bTerminateThread = TRUE;        /* Set End flag... -----*/
        /* ...and wait for end of thread -----*/
        WaitForSingleObject( hThread, INFINITE );
    }
    PostQuitMessage( 0 );                /* End of application */
    break;
}
return DefWindowProc( hWnd, uMsg, wP, lP );
}

/*****
/* WinMain : Start function of application */
/* -----*/
/* Parameter:      Default WinMain parameters */
/* Return value: Default WinMain return value */
/*****
int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR      lpCmdLine,
                    int         nCmdShow )
{
    WNDCLASS wc;    /* Each application is a process and thus */
                   /* requires its own Registry of the window class */

    wc.style          = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc    = WndProc;
    wc.cbClsExtra     = 0;
    wc.cbWndExtra     = 0;
    wc.hInstance      = NULL;
    wc.hIcon          = NULL;
    wc.hCursor        = LoadCursor( NULL, IDI_APPLICATION );
    wc.hbrBackground  = ( HBRUSH )(COLOR_BTNFACE + 1);
    wc.lpszMenuName    = MAKEINTRESOURCE( IDR_MENU );
    wc.lpszClassName  = "Pythagoras";
    if( RegisterClass( &wc ) )
    { /* Class successfully registered -----*/
        HWND hWnd = CreateWindow( "Pythagoras",          /* Create window */
                                "Pythagoras",
                                WS_OVERLAPPEDWINDOW,
                                0,0,300,300,
                                HWND_DESKTOP,
                                NULL, NULL, NULL );

        if( hWnd )
        { /* Window created? */
            MSG msg;

            ShowWindow( hWnd, nCmdShow );                /* Show window */
            UpdateWindow( hWnd );

            while( GetMessage( &msg, NULL, 0,0 ) ) /* Message loop */
            { /* ended by PostQuitMessage() */
                TranslateMessage( &msg );
                DispatchMessage( &msg );
            }
        }
    }
}

```

```
    }  
  }  
}  
return 0;  
}
```