

```

/*****
/*
/*----- S L A V E 3 2 . C -----*/
/* Task      : Slave processes that are controlled by a master.      */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich                    */
/* developed on :                                                        */
/* last update :                                                        */
/*****
#include <windows.h>
#include <math.h>

#include "turtle32.h"

#include "privat.h"

#include "resource.h"

/*= Global Variables =====*/

/* Tells the thread to terminate -----*/
BOOL      bTerminateThread;

/* Tells the thread to start drawing again -----*/
BOOL      bStartAgain;

/* Handle of thread -----*/
HANDLE     hThread;
HWND       hWndMaster;          /* Window handle of master application */

/* 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;          /* Delay after the completion of a cycle */
    HANDLE     hTerminateEvent; /* Kernel event for thread termination */
    HANDLE     hAccessEvent;    /* Kernel event for safe structure access */
} PYTHOPARMS;
typedef PYTHOPARMS *PPYTHOPARMS;

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

/*****
/* Pythagoras: Draw subtree of tree of Pythagoras */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/*                lEdge    : Edge length 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 < 4 ) return; /* subtree too small, so end of recursion */
    if( bTerminateThread ) return;
    if( 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 );
Pythagoras( pTC, ( long )(cos( ( PP.fAlpha * PI ) / 180.0 ) * Edge),
            bDraw ); /* draw left subtree */
turtlePop( pTC );

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

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

/*****
/* threadPainterFunction: Workhorse of PainterThread */
/* -----*/
/* Parameter: lpStart : not required */
/* Return value: Thread exit code */
/* -----*/
/* Info: This thread is created after entering the parameters in a */
/* dialog box, and paints the tree of Pythagoras continuously. */
*****/
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 color or 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;
}

/*****
/* 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 wP, LPARAM lP )
{
    switch( uMsg )
    {
        case WM_CREATE:
        {
            char Buffer[ 200 ];
            hThread = NULL; /* still no thread */
            /* Send the creation of the process and window handle of the */
            /* slave to the master. */
            SendMessage( hWndMaster, PM_PROCESSCREATED, 0, ( LONG )hWnd );
            wsprintf( Buffer, "Slave32 hWnd: 0x%X", hWnd );
            SetWindowText( hWnd, Buffer );
        }
        break;

        case PM_ANGLE:
            PP.fAlpha = ( float )lP; /* Receive angle from master */
            break;

        case PM_COLOR:
            PP.crColor = ( COLORREF )lP; /* Receive color from master */
            break;

        case PM_CREATETHREAD: /* Create thread */
        {
            DWORD dwThreadID;
            if( hThread ) /* terminate any existing thread */
            {
                bTerminateThread = TRUE; /* Set flag... */
                WaitForSingleObject( hThread, INFINITE ); /* wait for end */
            }

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

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

            bTerminateThread = FALSE; /* don't terminate thread */
            bStartAgain = FALSE; /* the tree will be repainted anyway */

            /* Create thread -----*/
            hThread = CreateThread( NULL,
                                   0L,
                                   threadPainterFunction,
                                   NULL,
                                   0L,
                                   &dwThreadID );
            if( hThread == NULL )
                MessageBox( hWnd,
                            "Could not create thread!",
                            "Error",
                            MB_OK );
        }
        break;

        case PM_EXITPROCESS: /* Master commands: Exit yourself! */
            DestroyWindow( hWnd );
            break;

        case WM_SIZE:
            bStartAgain = TRUE; /* Start drawing tree again */
            break;

        case WM_DESTROY: /* Close window, thus terminating thread */
            if( hThread )
            {
                bTerminateThread = TRUE; /* Set Exit flag... */
                WaitForSingleObject( hThread, INFINITE ); /* wait for end */
            }
            PostQuitMessage( 0 ); /* End of application */
            /* Famous Last Words: -----*/
            SendMessage( hWndMaster, PM_PROCESSHASEXIT, 0L, ( LONG )hWnd );
            break;
    }
}

```

```

    return DefWindowProc( hWnd, uMsg, wParam, lParam );
}

/*****
/* WinMain : Main entry point of application
/*-----*/
/* Parameter:      Default WinMain parameters
/* Return value: Default WinMain return value
*****/
int WINAPI WinMain( HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow )
{
    char Buffer[ 200 ];
    WNDCLASS wc; /* Each application is a process and thus requires
                  /* its own Registry of the window class

    /* Read environment variable created by the master thread

    if( !GetEnvironmentVariable( "Window", Buffer, sizeof( Buffer ) ) )
    {
        MessageBox( NULL, "Invalid environment variable", "Error", 0 );
        return 0;
    }

    hWndMaster = ( HWND )atoi( Buffer ); /* Handle of master window */

    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    = 0;
    wc.lpszClassName  = "Slave32";
    if( RegisterClass( &wc ) ) /* Class successfully registered? */
    {
        HWND hWnd = CreateWindow( "Slave32", /* Create window */
                                "Slave32",
                                WS_OVERLAPPEDWINDOW,
                                CW_USEDEFAULT,
                                CW_USEDEFAULT,
                                CW_USEDEFAULT,
                                CW_USEDEFAULT,
                                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 */
            {
                /* is ended by PostQuitMessage() */
                TranslateMessage( &msg );
                DispatchMessage( &msg );
            }
        }
        else MessageBox( NULL, "Window cannot be created!",
                        "Error", MB_OK );
    }
    return 0;
}

```