

```

/*****
/*
/*----- T U R T L E 3 2 . C -----*/
/* Task      : Implementation of a LOGO turtle. The turtle is
/*              controlled by a context structure.
/*-----*/
/* Author      : Michael Tischer and Bruno Jennrich
/* developed on :
/* last update  :
/*****/

#include <windows.h>
#include <math.h>
#include <memory.h>
#include <assert.h>

#include "turtle32.h"

/*== Macros for simplifying TLS access =====*/
#define GetX()          (pTC->lX)
#define GetY()          (pTC->lY)
#define GetAngle()      (pTC->fAngle)
#define GetLineWidth()  (pTC->lLineWidth)
#define GetColor()      (pTC->crColor)
#define GetPen()        (pTC->hPen)
#define GetWnd()        (pTC->hWnd)
#define GetStackMem()   (pTC->pStack)
#define GetStackPtr()   (pTC->lStackPtr)
#define GetBoundingLeft() (pTC->BR.left)
#define GetBoundingTop() (pTC->BR.top)
#define GetBoundingRight() (pTC->BR.right)
#define GetBoundingBottom() (pTC->BR.bottom)
#define GetUseBounding() (pTC->bUseBounding)

#define SetX(v)          pTC->lX=(v)
#define SetY(v)          pTC->lY=(v)
#define SetAngle(v)      pTC->fAngle=(v)
#define SetLineWidth(v)  pTC->lLineWidth=(v)
#define SetColor(v)      pTC->crColor=(v)
#define SetPen(v)        pTC->hPen=(v)
#define SetWnd(v)        pTC->hWnd=(v)
#define SetStackMem(v)   pTC->pStack=(v)
#define SetStackPtr(v)   pTC->lStackPtr=(v)
#define SetBoundingLeft(v) pTC->BR.left=(v)
#define SetBoundingTop(v) pTC->BR.top=(v)
#define SetBoundingRight(v) pTC->BR.right=(v)
#define SetBoundingBottom(v) pTC->BR.bottom=(v)
#define SetUseBounding(v) pTC->bUseBounding=(v)

/*== Maximum number of context items on the stack =====*/
#define MAX_TURTLESTACK 100

/*****
/* turtleInit: Put turtle in original state.
/*-----*/
/* Parameter:   pTC : TurtleContext
/* Return value: none
/*-----*/
/* Info: This function is automatically called when a new thread
/*        is started whose parent process links this
/*        DLL. Never call the function directly, because the
/*        stack memory is also allocated here.
/*****/
void WINAPI turtleInit( PTURTLECONTEXT pTC )
{
    PTURTLECONTEXT pStack;          /* Pointer for stack memory */

    SetX( 0 );
    SetY( 0 );
    SetAngle( ( float ) 0 );
    SetLineWidth( 0 );
    SetColor( RGB( 0,0,0 ) );
    SetPen( CreatePen( PS_SOLID, GetLineWidth(), GetColor() ) );
    SetWnd( 0 );
    SetUseBounding( FALSE );
    turtleInitBounding( pTC );

```

```

pStack = VirtualAlloc( NULL, /* Allocate stack */
                      sizeof( TURTLECONTEXT ) * MAX_TURTLESTACK,
                      MEM_COMMIT,
                      PAGE_READWRITE );
SetStackMem( pStack );
SetStackPtr( 0 );
}

/*****
/* turtleInitBounding: Initializes the bounding rectangle */
/*-----*/
/* Parameter:    pTC : TurtleContext */
/* Return value: none */
/*-----*/
/* Info: This function initializes the coordinates of the bounding
/* rectangle. The maximum and minimum drawing positions are
/* entered in this rectangle. That way you can implement a
/* "blind" passage with which you can first enter the required
/* drawing area in the bounding rectangle. You can use this
/* rectangle for subsequent drawing operations to scale the
/* display, so that the drawing always takes up the same
/* amount of space in the client area of a
/* window.
/* (see DLLMain)
*****/
void WINAPI turtleInitBounding( PTURTLECONTEXT pTC )
{
    SetBoundingLeft( 32767 ); /* minimum and maximum */
    SetBoundingTop( 32767 ); /* permissible coordinates */
    SetBoundingRight( -32768 );
    SetBoundingBottom( -32768 );
    SetUseBounding( FALSE );
}

/*****
/* turtleUseBounding: Sets flag for using BoundingRect */
/*-----*/
/* Parameter:    pTC : TurtleContext */
/*               bUse : True - Scale output to BoundingRect */
/*               False - Output to absolute MM_TEXT */
/*               coordinates */
/* Return value: none */
/*-----*/
/* Info: Bounding is used in the following way:
/* turtleInitBounding();
/* invisible Paint (turtleForward());
/* turtleUseBounding( TRUE );
/* Paint...
*****/
void WINAPI turtleUseBounding( PTURTLECONTEXT pTC, BOOL bUse )
{
    SetUseBounding( bUse );
}

/*****
/* turtleExit: Release memory and GDI objects. */
/*-----*/
/* Parameter:    pTC : TurtleContext */
/* Return value: none */
/*-----*/
/* Info: This function is automatically called whenever a thread
/* terminates. (see DLLMain)
*****/
void WINAPI turtleExit( PTURTLECONTEXT pTC )
{
    if( GetStackMem() ) /* does stack memory exist? */
        VirtualFree( GetStackMem(),
                     sizeof( TURTLECONTEXT ) * MAX_TURTLESTACK,
                     MEM_DECOMMIT );

    /* does pen exist ? */
    if( GetPen() ) DeleteObject( GetPen() );
}

/*****
/* turtleSetPen: Creates new current pen.
*****/

```

```

/*-----*/
/* Parameter:   pTC      : TurtleContext          */
/*              crCol    : Color of new pen        */
/*              lLineWidth: Width of pen           */
/* Return value: none                             */
/*****/
void WINAPI turtleSetPen( PTURTLECONTEXT pTC,
                        COLORREF crCol,
                        LONG lLineWidth )
{
    /* release any existing pen */
    if( GetPen() ) DeleteObject( GetPen() );

    /* place pen in TLS */
    SetPen( CreatePen( PS_SOLID, lLineWidth, crCol ) );
    SetColor( crCol );
    SetLineWidth( lLineWidth );
}

/*****/
/* turtleSetWindow: Set output window of thread. */
/*-----*/
/* Parameter:   pTC      : TurtleContext          */
/*              hWnd     : Handle of output window */
/* Return value: none                             */
/*-----*/
/* Info : Each turtle thread can manage only one output window.
/*         To display output in more than one window, you have to
/*         specify the output window using turtleSetWnd(). All the
/*         drawing commands that follow will have to do with this
/*         window.
/* Extension : Before a GDI command is executed in turtleForward(),
/*              the DC is initialized with the required objects.
/*              Each time the pen to be used is selected and the
/*              scaling mode (BoundingRect) is set. However,
/*              this is quite inefficient. On the other hand,
/*              by using a window with the CS_OWNDC style, the DC
/*              only needs to be updated when the pen or the
/*              drawing area is changed. Windows saves the state
/*              of the DCs for such windows internally, so that
/*              GetDC() returns the calls of a preinitialized
/*              DC.
/*****/
void WINAPI turtleSetWindow( PTURTLECONTEXT pTC, HWND hWnd )
{
    SetWnd( hWnd );
}

/*****/
/* turtleRotate: Change turtle's orientation or direction of movement */
/*-----*/
/* Parameter:   pTC      : TurtleContext          */
/*              fAngle   : Angle by which turtle is to be rotated,
/*                        in degrees. Positive values rotate counter-
/*                        clockwise
/* Return value: none
/*****/
void WINAPI turtleRotate( PTURTLECONTEXT pTC, float fAngle )
{
    /* Limit angle to values between 0 and 360 */
    SetAngle( ( float )fmod( GetAngle() + (360.0 - fAngle) , 360.0 ) );
}

/*****/
/* turtleSetAngle: Set absolute orientation */
/*-----*/
/* Parameter:   fAngle - Angle at which the turtle is to "look",
/*              in degrees.
/* Return value: none
/*****/
void WINAPI turtleSetAngle( PTURTLECONTEXT pTC, float fAngle )
{
    SetAngle( ( float )fmod( fAngle + 360.0, 360.0 ) );
}

/*****/
/* turtleForward: Move turtle */

```

```

/*-----*/
/* Parameter:      pTC      : TurtleContext      */
/*                fLineLen  : Length of distance to be covered      */
/*                bDraw     : should a line be drawn during          */
/*                          movement?                      */
/* Return value: none      */
/******/
void WINAPI turtleForward( PTURTLECONTEXT pTC,
                          float fLineLen,
                          BOOL bDraw )
{
    double dRadiant;
    POINT pStart;
    HDC hDC = GetWnd() ? GetDC( GetWnd() ) : 0;

    assert( hDC );

    /* degrees to radiant */
    dRadiant = ( GetAngle() / 180.0 ) * 3.141592654;
    /* If a line needs to be drawn, the starting point of the line ---*/
    /* must be saved. ---*/
    if( ( bDraw ) && ( hDC ) )
    {
        pStart.x = GetX();
        pStart.y = GetY();
    }

    /* Calculate new position based on the current angle -----*/
    SetX( GetX() + ( int )( cos( dRadiant ) * fLineLen ) );
    SetY( GetY() + ( int )( sin( dRadiant ) * fLineLen ) );

    /* Update BoundingRect -----*/
    if( GetX() < GetBoundingLeft() ) SetBoundingLeft( GetX() );
    if( GetX() > GetBoundingRight() ) SetBoundingRight( GetX() );
    if( GetY() < GetBoundingTop() ) SetBoundingTop( GetY() );
    if( GetY() > GetBoundingBottom() ) SetBoundingBottom( GetY() );

    if( ( bDraw ) && ( hDC ) ) /* Draw line? */
    {
        /* Can be refined. (see turtleSetWnd() and CS_OWNDC ) -----*/
        HPEN hOldPen = SelectObject( hDC, GetPen() );
        int iMap;

        if( GetUseBounding() ) /* Use BoundingRect? */
        {
            RECT r;

            /* Map all coordinates of BoundingRect to ClientArea of -----*/
            /* window. -----*/
            GetClientRect( GetWnd(), &r );
            /* Distort scaling -----*/
            iMap = SetMapMode( hDC, MM_ANISOTROPIC );

            /* logical coordinates = BoundingRect -----*/
            SetWindowOrgEx( hDC, GetBoundingLeft(), GetBoundingTop(), NULL );
            SetWindowExtEx( hDC, ( GetBoundingRight() - GetBoundingLeft() ),
                          ( GetBoundingBottom() - GetBoundingTop() ),
                          NULL );

            /* map to ClientArea -----*/
            SetViewportOrgEx( hDC, 0, 0, NULL );
            SetViewportExtEx( hDC, ( r.right - r.left ),
                          ( r.bottom - r.top ),
                          NULL );
        }

        MoveToEx( hDC, pStart.x, pStart.y, NULL ); /* Draw line */
        LineTo( hDC, GetX(), GetY() );

        SelectObject( hDC, hOldPen ); /* select old pen */

        if( GetUseBounding() ) /* restore old drawing mode */
            SetMapMode( hDC, iMap );
    }
    if( hDC ) ReleaseDC( GetWnd(), hDC );
}

/******/

```

```

/* turtleMoveTo: Set current drawing position */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/*                lX       : new X coordinate */
/*                lY       : new Y coordinate */
/*                bDraw    : draw a line at the new */
/*                          point? */
/* Return value: none */
/*-----*/
void WINAPI turtleMoveTo( PTURTLECONTEXT pTC,
                          LONG lX,
                          LONG lY,
                          BOOL bDraw )
{
    POINT pStart;
    HDC hDC = GetWnd() ? GetDC( GetWnd() ) : 0;

    assert( hDC );

    /* If a line needs to be drawn, the starting point of the line ---*/
    /* must be saved. ---*/
    if( ( bDraw ) && ( hDC ) )
    {
        pStart.x = GetX();
        pStart.y = GetY();
    }

    SetX( lX ); /* Set new position */
    SetY( lY );

    /* Update BoundingRect -----*/
    if( GetX() < GetBoundingLeft() ) SetBoundingLeft( GetX() );
    if( GetX() > GetBoundingRight() ) SetBoundingRight( GetX() );
    if( GetY() < GetBoundingTop() ) SetBoundingTop( GetY() );
    if( GetY() > GetBoundingBottom() ) SetBoundingBottom( GetY() );

    if( ( bDraw ) && ( hDC ) ) /* Draw line? */
    {
        /* Can be refined. (see turtleSetWnd() and CS_OWNDC ) -----*/
        HPEN holdPen = SelectObject( hDC, GetPen() );
        int iMap;

        if( GetUseBounding() ) /* Use BoundingRect? */
        {
            RECT r;

            /* Map all coordinates of BoundingRect to ClientArea of -----*/
            /* window. -----*/
            GetClientRect( GetWnd(), &r );
            /* Distort scaling -----*/
            iMap = SetMapMode( hDC, MM_ANISOTROPIC );

            /* logical coordinates = BoundingRect -----*/
            SetWindowOrgEx( hDC, GetBoundingLeft(), GetBoundingTop(), NULL );
            SetWindowExtEx( hDC, ( GetBoundingRight() - GetBoundingLeft() ),
                           ( GetBoundingBottom() - GetBoundingTop() ),
                           NULL );

            /* map to ClientArea -----*/
            SetViewportOrgEx( hDC, 0, 0, NULL );
            SetViewportExtEx( hDC, (r.right - r.left),
                             (r.bottom - r.top),
                             NULL );
        }

        MoveToEx( hDC, pStart.x, pStart.y, NULL ); /* Draw line */
        LineTo( hDC, GetX(), GetY() );

        SelectObject( hDC, holdPen ); /* select old pen */

        if( GetUseBounding() ) /* restore old drawing mode */
            SetMapMode( hDC, iMap );
    }
    if( hDC ) ReleaseDC( GetWnd(), hDC );
}

```

```

/*****
/* turtleCircle : Draw a circle at current position */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/*                fRadius  : Radius of circle */
/*                bDraw   : draw circle? */
/* Return value: none */
/*****
void WINAPI turtleCircle( PTURTLECONTEXT pTC,
                        float fRadius,
                        BOOL bDraw )
{
    HDC hDC = GetWnd() ? GetDC( GetWnd() ) : 0;

    assert( hDC );

    /* Update BoundingRect -----*/
    if( GetX() - fRadius < GetBoundingLeft() )
        SetBoundingLeft( GetX() - ( long )fRadius );
    if( GetX() + fRadius > GetBoundingRight() )
        SetBoundingRight( GetX() + ( long )fRadius );
    if( GetY() - fRadius < GetBoundingTop() )
        SetBoundingTop( GetY() - ( long )fRadius );
    if( GetY() + fRadius > GetBoundingBottom() )
        SetBoundingBottom( GetY() + ( long )fRadius );

    if( ( bDraw ) && ( hDC ) )                /* Draw line? */
    {
        /* Can be refined. (see turtleSetWnd() and CS_OWNDC ) -----*/
        HPEN hOldPen = SelectObject( hDC, GetPen() );
        HBRUSH hOldBrush = SelectObject( hDC, GetStockObject( NULL_BRUSH ) );
        int iMap;

        if( GetUseBounding() )                /* Use BoundingRect? */
        {
            RECT r;

            /* Map all coordinates of BoundingRect to ClientArea of -----*/
            /* window. -----*/
            GetClientRect( GetWnd(), &r );
            /* Distort scaling -----*/
            iMap = SetMapMode( hDC, MM_ANISOTROPIC );

            /* logical coordinates = BoundingRect -----*/
            SetWindowOrgEx( hDC, GetBoundingLeft(), GetBoundingTop(), NULL );
            SetWindowExtEx( hDC, ( GetBoundingRight() - GetBoundingLeft() ),
                            ( GetBoundingBottom() - GetBoundingTop() ),
                            NULL );

            /* map to ClientArea -----*/
            SetViewportOrgEx( hDC, 0, 0, NULL );
            SetViewportExtEx( hDC, (r.right - r.left),
                              (r.bottom - r.top),
                              NULL );
        }

        Ellipse( hDC, GetX() - ( long )fRadius, GetY() - ( long )fRadius,
                 GetX() + ( long )fRadius, GetY() + ( long )fRadius );

        SelectObject( hDC, hOldPen );                /* select old pen */
        SelectObject( hDC, hOldBrush );              /* select old brush */

        if( GetUseBounding() )                /* restore old drawing mode */
            SetMapMode( hDC, iMap );
    }
    if( hDC ) ReleaseDC( GetWnd(), hDC );
}

/*****
/* turtleCircle : Output text at current position */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/*                lpText   : Address of text buffer */
/* Return value: none */
/*****
void WINAPI turtleTextOut( PTURTLECONTEXT pTC,
                        LPSTR lpText )

```

```

{
    HDC hDC = GetWnd() ? GetDC( GetWnd() ) : 0;
    assert( hDC );
    if( hDC )
    {
        /* Draw line? */
        int iMap;
        int iAlign;
        COLORREF crBkColor;

        if( GetUseBounding() )
        {
            /* Use BoundingRect? */
            RECT r;
            /* Map all coordinates of BoundingRect to ClientArea of -----*/
            /* window. -----*/
            GetClientRect( GetWnd(), &r );
            /* Distort scaling -----*/
            iMap = SetMapMode( hDC, MM_ANISOTROPIC );

            /* logical coordinates = BoundingRect -----*/
            SetWindowOrgEx( hDC, GetBoundingLeft(), GetBoundingTop(), NULL );
            SetWindowExtEx( hDC, ( GetBoundingRight() - GetBoundingLeft() ),
                            ( GetBoundingBottom() - GetBoundingTop() ),
                            NULL );

            /* map to ClientArea -----*/
            SetViewportOrgEx( hDC, 0, 0, NULL );
            SetViewportExtEx( hDC, (r.right - r.left),
                              (r.bottom - r.top),
                              NULL );
        }

        iAlign = SetTextAlign( hDC, TA_CENTER | TA_TOP );
        crBkColor = SetBkColor( hDC, GetSysColor( COLOR_3DFACE ) );
        TextOut( hDC, GetX(), GetY(), lpText, lstrlen( lpText ) );
        SetBkColor( hDC, crBkColor );
        SetTextAlign( hDC, iAlign );

        if( GetUseBounding() )
            SetMapMode( hDC, iMap );
    }
    if( hDC ) ReleaseDC( GetWnd(), hDC );
}

/*****
/* turtlePush: Push current turtle state onto custom TurtleStack */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/* Return value: TRUE  - Able to push context */
/*               FALSE - Stack overflow */
*****/
BOOL WINAPI turtlePush( PTURTLECONTEXT pTC )
{
    assert( GetStackPtr() < MAX_TURTLESTACK );

    if( GetStackPtr() < MAX_TURTLESTACK )
    {
        PTURTLECONTEXT pStack;
        pStack = GetStackMem();
        memcpy( &pStack[ GetStackPtr() ], pTC, sizeof( TURTLECONTEXT ) - NOCTXSIZE );
        /* Save context */
        SetStackPtr( GetStackPtr() + 1 );
        /* increment stack pointer */
        return TRUE;
    }
    return FALSE;
}

/*****
/* turtlePop: Pop current turtle state from stack */
/*-----*/
/* Parameter:      pTC      : TurtleContext */
/* Return value: TRUE  - Able to pop context */
/*               FALSE - Stack underflow */
*****/
BOOL WINAPI turtlePop( PTURTLECONTEXT pTC )
{
    assert( GetStackPtr() > 0 );
}

```

```

if( GetStackPtr() > 0 )
{
    PTURTLECONTEXT pStack;
    pStack = GetStackMem();
    SetStackPtr( GetStackPtr() - 1 );          /* decrement stack pointer */
    /* Restore context -----*/
    memcpy( pTC, &pStack[ GetStackPtr() ], sizeof( TURTLECONTEXT ) - NOCTXTSIZE );
    return TRUE;
}
return FALSE;
}

```