

```

/*****
*
*          S 3 2 4 0 C . C
*
**-----**
* Task          : Demonstrates sprites in 320x400 VGA graphic
*                mode using 256 colors and two screen pages.
*                This program requires the assembly language
*                modules V3240CA.ASM and S3240CA.ASM.
**-----**
* Author        : Michael Tischer
* Developed on   : 09/09/90
* Last update on : 02/27/92
**-----**
* Memory model   : SMALL
**-----**
* (MICROSOFT C)
* Compilation    : CL /AS /c /W0 s3240c.c
*                 LINK s3240c v3240ca s3240ca;
**-----**
* (BORLAND TURBO C)
* Compilation    : Create a project file containing the following:
*                 s3240c.c
*                 v3240ca.asm
*                 s3240ca.asm
**-----**
* Call          : s3240c
*****/

```

```

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

```

```

/*-- Compiler dependent declarations -----*/

```

```

#ifdef __TURBOC__
    #include <alloc.h>
#else
    #include <malloc.h>
    #define random(x) ( rand() % (x+1) )          /* Random function */
#endif

```

```

/*-- Type declarations -----*/

```

```

typedef unsigned char BYTE;
typedef BYTE BOOL;

```

```

typedef struct {
    /* Pixel buffer for GetVideo() and PutVideo() */
    BYTE *bitptr[4],          /* Pointer to bitplanes */
        byprod[4],          /* Number of bytes to be copied */
        rhght;              /* Number of rows */
    /*-- Here follow bytes from the bitplanes -----*/
} PIXBUF;

```

```

typedef PIXBUF *PIXPTR;          /* Pointer to a pixel buffer */

```

```

typedef struct {
    /* Sprite design */
    BYTE twidth,              /* Total width */
        theight;             /* Height in pixel lines */
    PIXPTR pixbp;             /* Pointer to pixel block */
} SPLOOK;

```

```

typedef struct {
    /* Sprite descriptor (ID) */
    SPLOOK *splookp;          /* Pointer to sprite design */
    int x[2], y[2];           /* Coordinates: pp 0&1 */
    PIXPTR hgptr[2];          /* Pointer to background buffer */
} SPID;

```

```

/*-- External references to assembler routines -----*/

```

```

extern void init320400( void );
extern void setpix( int x, int y, unsigned char pcolor);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

```

```

extern void copybuf2plane( BYTE *bufptr, BYTE page,
                          int tox, int toy, BYTE rwidth,
                          BYTE rheight, BOOL bg );
extern void copyplane2buf( BYTE *bufptr, BYTE page,
                          int fromx, int fromy, BYTE rwidth,
                          BYTE rheight );

/*-- Constants -----*/

#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

#define MAXX 319          /* Maximum X- and Y-coordinates */
#define MAXY 399

#define OUT_LEFT  1 /* For collision documentation in SpriteMove() */
#define OUT_TOP   2
#define OUT_RIGHT  4
#define OUT_BOTTOM 8
#define OUT_NO     0 /* None */

#define ALLOCBUF ((PIXPTR) 0) /* GetVideo(): Allocate buffer */

/*****
 * IsVga : Determines whether a VGA card is installed.
 *****/
*-----*
* Input   : None
* Output  : 0 if no VGA card exists, otherwise <> 0
*****/

BYTE IsVga( void )
{
    union REGS Regs; /* Processor registers for interrupt call */

    Regs.x.ax = 0x1a00; /* Function 1AH applies to VGA only */
    int86( 0x10, &Regs, &Regs );
    return (BYTE) ( Regs.h.al == 0x1a ); /* Is the function available? */
}

/*****
 * Line: Draws a line based on the Bresenham algorithm.
 *****/
*-----*
* Input   : X1, Y1 = Starting coordinates (0 - ...)
*          : X2, Y2 = Ending coordinates
*          : LPCOL = Color of line pixels
*****/

/*-- Function for swapping two integer variables -----*/

void SwapInt( int *i1, int *i2 )
{
    int dummy;

    dummy = *i2;
    *i2 = *i1;
    *i1 = dummy;
}

/*-- Main function -----*/

void Line( int x1, int y1, int x2, int y2, BYTE lpcol )
{
    int d, dx, dy,
        aincr, bincr,
        xincr, yincr,
        x, y;

    if ( abs(x2-x1) < abs(y2-y1) ) /* X- or Y-axis overflow? */
    { /* Check Y-axes */
        if ( y1 > y2 ) /* y1 > y2? */
        {
            SwapInt( &x1, &x2 ); /* Yes --> Swap X1 with X2 */
            SwapInt( &y1, &y2 ); /* and Y1 with Y2 */
        }
    }
}

```

```

xincr = ( x2 > x1 ) ? 1 : -1;          /* Set X-axis increment */

dy = y2 - y1;
dx = abs( x2-x1 );
d = 2 * dx - dy;
aincr = 2 * (dx - dy);
bincr = 2 * dx;
x = x1;
y = y1;

setpix( x, y, lpcol );                  /* Set first pixel */
for (y=y1+1; y<= y2; ++y )              /* Execute line on Y-axes */
{
    if ( d >= 0 )
    {
        x += xincr;
        d += aincr;
    }
    else
        d += bincr;
    setpix(x, y, lpcol);
}
else                                     /* Check X-axes */
{
    if ( x1 > x2 )                       /* x1 > x2? */
    {
        SwapInt( &x1, &x2 );             /* Yes --> Swap X1 with X2 */
        SwapInt( &y1, &y2 );             /* and Y1 with Y2 */
    }

    yincr = ( y2 > y1 ) ? 1 : -1;        /* Set X-axis increment */

    dx = x2 - x1;
    dy = abs( y2-y1 );
    d = 2 * dy - dx;
    aincr = 2 * (dy - dx);
    bincr = 2 * dy;
    x = x1;
    y = y1;

    setpix(x, y, lpcol);                 /* Set first pixel */
    for (x=x1+1; x<=x2; ++x )            /* Execute line on X-axes */
    {
        if ( d >= 0 )
        {
            y += yincr;
            d += aincr;
        }
        else
            d += bincr;
        setpix(x, y, lpcol);
    }
}

/*****
* PrintChar : Writes a character to the screen while in graphic mode.*
**-----**
* Input      : THECHAR = Character to be written                      *
*              x, y    = X- and Y-coordinates of upper-left corner    *
*              FG      = Foreground color                             *
*              BK      = Background color                             *
* Info       : Character is created in an 8x8 matrix, based on the    *
*              8x8 ROM font.                                           *
*****/

void PrintChar( char thechar, int x, int y, BYTE fg, BYTE bk )
{
    typedef BYTE FDEF[256][8];          /* Font array */
    typedef FDEF far *TPTR;              /* Pointer to font */

    BYTE i, k,                          /* Loop counter */
          BMask;                        /* Bit mask for character design */

    static TPTR fptr = (TPTR) 0;        /* Pointer to font in ROM */

```

```

if ( fptr == (TPTR) 0 )          /* Pointer to font already set? */
    fptr = getfontptr(); /* No --> Use the assembler function to load */

/*- Generate character pixel by pixel -----*/

if ( bk == 255 )                /* Drawing transparent characters? */
    for ( i = 0; i < 8; ++i )    /* Yes --> Set foreground pixels only */
    {
        BMask = (*fptr)[thechar][i]; /* Get bit pattern for one line */
        for ( k = 0; k < 8; ++k, BMask <= 1 ) /* Execute columns */
            if ( BMask & 128 ) /* Pixel set? */
                setpix( x+k, y+i, fg ); /* Yes */
    }
else                             /* No --> Consider background */
    for ( i = 0; i < 8; ++i )    /* Execute lines */
    {
        BMask = (*fptr)[thechar][i]; /* Get bit pattern for one line */
        for ( k = 0; k < 8; ++k, BMask <= 1 ) /* Execute columns */
            setpix( x+k, y+i, ( BMask & 128 ) ? fg : bk );
    }
}

```

```

/*****
* GrfxPrintf: Displays a formatted string on the graphic screen.
* Input      : X, Y      = Starting coordinates (0 - ...)
*              FG        = Foreground color
*              BK        = Background color (255 = transparent)
*              STRING    = String with format information
*              ...       = Arguments similar to printf
*****/

void GrfxPrintf( int x, int y, BYTE fg, BYTE bk, char * string, ... )
{
    va_list parameter;          /* Parameter list for VA_... macros */
    char stngbuf[255],          /* Buffer for formatted string */
        *cp;

    va_start( parameter, string );          /* Convert parameter */
    vsprintf( stngbuf, string, parameter ); /* Format */
    for ( cp = stngbuf; *cp; ++cp, x+= 8 ) /* Formatted string */
        PrintChar( *cp, x, y, fg, bk );    /* Display using PrintChar */
}

/*****
* GetVideo: Places contents of a rectangular range of video RAM
*           in a buffer.
*-----
* Input      : PAGE      = Screen page (0 or 1)
*              X1, Y1    = Starting coordinates
*              WRANGE    = Width of rectangular range in pixels
*              HRANGE    = Height of rectangular range in pixels
*              BUFPTR    = Pointer to pixel buffer into which
*                       information should be allocated
* Output     : Pointer to allocated pixel buffer with contents of
*             specified range
* Info       : If the BUFPTR parameter contains nothing, a new pixel
*             buffer is allocated and returned on the heap. This buffer
*             can be re-created on each call, provided the previous
*             contents are no longer needed, and provided the size
*             of the rectangular range remains unchanged.
*****/

PIXPTR GetVideo( BYTE page, int x1, int y1, BYTE wrange, BYTE hrange,
                PIXPTR bufptr )
{
    BYTE i,                                /* Loop counter */
        curplane,                          /* Currently processed bitplane */
        sb,                                /* Bitplane at starting coordinates */
        eb,                                /* Bitplane at ending coordinates */
        b,                                /* Number of bytes per bitplane */
        am;                               /* Number of bytes in center of both groups of four */
    BYTE *rptr;                            /* Pointer for bitplane position in buffer */

    if ( bufptr == ALLOCBUF )               /* No buffer passed on call? */
        bufptr = malloc( sizeof( PIXBUF ) + wrange*hrange ); /* No-> Alloc */

    /*-- Compute number of bytes per bitplane -----*/

    am = (BYTE) ( ( (x1+wrange-1) & ~3 ) - /* Number of bytes in center */
                ( (x1+4) & ~3 ) ) >> 2;
    sb = (BYTE) (x1 % 4);                  /* Starting bitplane */
    eb = (BYTE) ((x1+wrange-1) % 4);       /* Ending bitplane */

    rptr = (BYTE *) bufptr + sizeof( PIXBUF );

    /*-- Execute four bitplanes -----*/

    for ( i=0; i<4; ++i )
    {
        curplane = (sb+i) % 4;
        b = am;                                /* Base number of bytes to be copied */
        if ( curplane >= sb )                  /* Also in starting block of four? */
            ++b;                              /* Yes --> Add a byte to this bitplane */
        if ( curplane <= eb )                  /* Also in ending block of four? */
            ++b;                              /* Yes --> Add a byte to this bitplane */
        bufptr->bitptr[i] = rptr;              /* Place pointer at start of buffer */
        bufptr->byprod[i] = b;                /* Place number in buffer */
    }
}

```

```

        copyplane2buf( rptr, page, x1+i,                /* Get contents */
                        y1, b, hrange );                /* of bitplane */
        rptr += (b * hrange);                          /* Set pointer to next */
    };                                                  /* bitplane in buffer */

    bufptr->rhght = hrange;                            /* Store height */

    return bufptr;                                     /* Return buffer pointer to caller */
}

/*****
* PutVideo: Writes contents of a rectangular screen range generated
*           by GetVideo to video RAM.
**-----**
* Input    : BUFPTR = Pointer to pixel buffer from GetVideo
*           PAGE    = Screen page (0 or 1)
*           X1, Y1  = Starting coordinates
*           BG      = Should background pixels (color code 255) not
*                   be written to video RAM
* Info     : Pixel buffer is not cleared by this procedure; use the
*           FreePixBuf for this purpose.
*****/

void PutVideo( PIXPTR bufptr, BYTE page, int x1, int y1, BOOL bg )
{
    BYTE curplane,                                /* Currently executed bitplane */
        hrange;

    hrange = bufptr->rhght;                        /* Range height */
    for ( curplane=0; curplane<4; ++curplane ) /* Execute four bitplanes */
        copybuf2plane( bufptr->bitptr[curplane], page, x1+curplane,
                        y1, bufptr->byprod[curplane], hrange, bg );
}

/*****
* FreePixBuf: Clears a pixel buffer previously allocated by Heap.
**-----**
* Input    : BUFPTR = Pointer to pixel buffer
*****/

void FreePixBuf( PIXPTR bufptr )
{
    free( bufptr );
}

/*****
* CreateSprite: Creates a sprite based on a user-defined
*              pixel pattern.
**-----**
* Input    : SPLOOKP = Pointer to data structure from CompileSprite
* Output   : Pointer to created sprite structure
*****/

SPID *CreateSprite( SPLOOK *splookp )
{
    SPID *spidp;                                /* Pointer to created sprite structure */

    spidp = (SPID *) malloc( sizeof(SPID) ); /* Allocate spr. structure */
    spidp->splookp = splookp;                /* Pass data to the */
                                            /* sprite structure */

    /*- Create two background buffers for storing range from video RAM -*/

    spidp->hgptr[0] = GetVideo( 0, 0, 0, splookp->twidth,
                                splookp->theight, ALLOCBUF );
    spidp->hgptr[1] = GetVideo( 0, 0, 0, splookp->twidth,
                                splookp->theight, ALLOCBUF );
    return spidp;                               /* Return pointer to sprite structure */
}

/*****
* CompileSprite: Creates a sprite's pixel and bit patterns, based on
*               the sprite's definition at runtime.
**-----**
* Input    : BUFP    = Pointer to array containing string pointers
*               controlling sprite's pattern
*****/

```

```

*           SHEIGHT = Sprite height (and number of strings needed)  *
*           GPAGE    = Graphic page for sprite design                *
*           FB       = ASCII character for the smallest color        *
*           FGCOLOR  = First color code for FB                       *
* Info       : Sprite structure of pixel lines starts at left margin. *
*****

```

```

SPLOOK *CompileSprite( char **bufp, BYTE sheight, BYTE gpage,
                      char fb, BYTE fgcolor )
{
    BYTE    swidth,                      /* String width */
           c,                          /* Get character from c sprite array */
           i, k;                       /* Loop variables */
    SPLOOK *splookp;                   /* Pointer to create sprite structure */
    PIXPTR pbpPtr;                    /* Get temporary sprite background */

    /*-- Create SpriteLook structure and fill with data -----*/

    splookp = (SPLOOK *) malloc( sizeof(SPLOOK) );
    swidth = (BYTE) strlen( *bufp );
    splookp->twidht = swidth;
    splookp->theight = sheight;

    /*-- Place sprite in page at 0/0 -----*/

    setpage( gpage );                  /* Set page for drawing */
    showpage( gpage );
    pbpPtr = GetVideo( gpage, 0, 0, swidth, sheight, ALLOCBUF ); /* Bkgd. */

    for ( i = 0; i < sheight; ++i )    /* Execute rows */
        for ( k = 0; k < swidth; ++k ) /* Execute columns */
        {
            c = (*(bufp+i)+k);
            setpix( k, i, (BYTE) (c == ' ' ? 255 : fgcolor+(c-fb)));
        }

    /*-- Get sprite in buffer and restore background -----*/

    splookp->pixbp = GetVideo( gpage, 0, 0, swidth, sheight, ALLOCBUF );
    PutVideo( pbpPtr, gpage, 0, 0, FALSE );
    FreePixBuf( pbpPtr );              /* Free buffer */

    return splookp;                   /* Return pointer to sprite buffer */
}

```

```

/*****
* PrintSprite : Displays sprite in a specified page.                *
*****
* Input       : SPIDP      = Pointer to the sprite structure        *
*               SPRPAGE    = Page in which sprite should be drawn (0 or 1) *
*****

```

```

void PrintSprite( register SPID *spidp, BYTE sprpage )
{
    PutVideo( spidp->splookp->pixbp,
              sprpage, spidp->x[sprpage], spidp->y[sprpage], TRUE );
}

```

```

/*****
* GetSpriteBg: Gets a sprite background and specifies the position. *
*****
* Input       : SPIDP      = Pointer to the sprite structure        *
*               SPRPAGE    = Page from which the background should be taken *
*               (0 or 1)                                           *
*****

```

```

void GetSpriteBg( register SPID *spidp, BYTE sprpage )
{
    GetVideo( sprpage, spidp->x[sprpage], spidp->y[sprpage],
              spidp->splookp->twidht, spidp->splookp->theight,
              spidp->hgptr[sprpage] );
}

```

```

/*****
* RestoreSpriteBg: Restores sprite background from original graphic *
*                  page.                                           *

```

```

**-----**
*   Input   : SPIDP = Pointer to the sprite structure *
*             SPRPAGE = Page from which background should be copied *
*             (0 or 1) *
*****/

void RestoreSpriteBg( register SPID *spidp, BYTE sprpage )
{
    PutVideo( spidp->hgptr[sprpage], sprpage,
              spidp->x[sprpage], spidp->y[sprpage], FALSE );
}

/*****
*   MoveSprite: Copy sprite within background to original graphic page.*
**-----**
*   Input   : SPIDP = Pointer to the sprite structure *
*             SPRPAGE= Page to which the background should be copied *
*             (0 or 1) *
*             DELTAX = Movement counter in X- *
*             DELTAY  and Y-directions *
*   Output  : Collision marker (see OUT_ constants) *
*****/

BYTE MoveSprite( SPID *spidp, BYTE sprpage, int deltax, int deltay )
{
    int  newx, newy; /* New sprite coordinates */
    BYTE out; /* Display collision with border */

    /*-- Move X-coordinates and test for border collision -----*/

    if ( ( newx = spidp->x[sprpage] + deltax ) < 0 )
    {
        newx = 0 - deltax - spidp->x[sprpage];
        out = OUT_LEFT;
    }
    else
    {
        if ( newx > MAXX - spidp->splookp->twidth )
        {
            newx = (2*(MAXX+1))-newx-2*(spidp->splookp->twidth);
            out = OUT_RIGHT;
        }
        else
            out = OUT_NO;
    }

    /*-- Move Y-coordinates and test for border collision -----*/

    if ( ( newy = spidp->y[sprpage] + deltay ) < 0 ) /* Top border? */
    {
        /* Yes --> Deltay must be negative */
        newy = 0 - deltay - spidp->y[sprpage];
        out |= OUT_TOP;
    }
    else
    {
        if ( newy + spidp->splookp->theight > MAXY+1 ) /* Bottom? */
        {
            /* Yes --> Deltay must be positive */
            newy = (2*(MAXY+1))-newy-2*(spidp->splookp->theight);
            out |= OUT_BOTTOM;
        }
    }

    /*-- Set new position only if different from old position -----*/

    if ( newx != spidp->x[sprpage] || newy != spidp->y[sprpage] )
    {
        /* If there's a new position */
        RestoreSpriteBg( spidp, sprpage ); /* reset background and store */
        spidp->x[sprpage] = newx; /* new coordinates */
        spidp->y[sprpage] = newy;
        GetSpriteBg( spidp, sprpage ); /* Get new background */
        PrintSprite( spidp, sprpage ); /* Draw sprite in specified page */
    }
    return out;
}

/*****
*   SetSprite: Sets sprite at a specific position. *
**-----**
*   Input   : SPIDP = Pointer to the sprite structure *
*             x0, y0 = Sprite coordinates for page 0 *
*****/

```



```

*          x1, y1 = Sprite coordinates for page 1          *
* Info      : This function call should be made the first time that *
*          MoveSprite() is called. *
*****/

void SetSprite( SPID *spidp, int x0, int y0, int x1, int y1 )
{
    spidp->x[0] = x0;          /* Store coordinates in sprite structure */
    spidp->x[1] = x1;
    spidp->y[0] = y0;
    spidp->y[1] = y1;

    GetSpriteBg( spidp, 0 );          /* Get sprite backgrounds */
    GetSpriteBg( spidp, 1 );          /* in pages 0 and 1 */
    PrintSprite( spidp, 0 );          /* Draw sprite in */
    PrintSprite( spidp, 1 );          /* pages 0 and 1 */
}

/*****
* RemoveSprite: Removes a sprite from its current position and makes *
*          it invisible. *
**-----**
* Input      : SPIDP = Pointer to the sprite structure *
* Info       : After calling this function the SetSprite() function *
*          must be called before the sprite can be moved using the *
*          MoveSprite() function. *
*****/

void RemoveSprite( SPID *spidp )
{
    RestoreSpriteBg( spidp, 0 );          /* Reset sprite backgrounds */
    RestoreSpriteBg( spidp, 1 );          /* in pages 0 and 1 */
}

/*****
* Demo: Demonstrates these functions. *
*****/

void Demo( void )
{
    static char *StarShipUp[20] =
    {
        "          AA          ",
        "          AAAA          ",
        "          AAAA          ",
        "          AA          ",
        "          GGBBGG          ",
        "          GBBCCBBG          ",
        "          GBBBCCBBBG          ",
        "          GBBBBBBBBBBG          ",
        "          GBBBBBBBBBBG          ",
        " G          GBBBBBBBBBBBBG          G ",
        "GCG          GGBBBBBBBBBBBDGG          GCG",
        "GCG          GGBBBDDBB          BBDBBBBG          GCG",
        "GCBGGGBBBBBBDBB          BBDBBBBGGGBCG",
        "GCBBBBBBBBBBDB          BDBBBBBBBBBBCG",
        "BBBBBBBBBBBBBDB          BB          BDBBBBBBBBBBBB",
        "GGCBBBBBBDBBBBBBBBBBDBBBBBBBBCG",
        "          GGCCBBDDDDDDDDDDDDDBBBCCG",
        "          GGBDDDDDGGGGGDDDDDBBG",
        "          GDDDDGGG          GGGDDDDG",
        "          DDDD          DDDD          " };

    static char *StarShipDown[20] =
    {
        "          DDDD          DDDD          ",
        "          GDDDDGGG          GGGDDDDG",
        "          GGBDDDDDGGGGGDDDDDBBG",
        "          GGCCBBDDDDDDDDDDDDDBBBCCG",
        "GGCBBBBBBDBBBBBBBBBBDBBBBBBBBCG",
        "BBBBBBBBBBBBBDB          BB          BDBBBBBBBBBBBB",
        "GCBBBBBBBBBBDB          BDBBBBBBBBBBCG",
        "GCBGGGBBBBBBDBB          BBDBBBBGGGBCG",
        "GCG          GGBBBDDBB          BBDBBBBG          GCG",
        "GCG          GGBBBBBBBBBBBDGG          GCG",
        " G          GBBBBBBBBBBBBBG          G " };

```

```

        "          GBBBBBBBBBBG          " ,
        "          GBBBBBBBBBBG          " ,
        "          GBBBCCBBBG          " ,
        "          GBBCCBBG          " ,
        "          GGBBGG          " ,
        "          AA          " ,
        "          AAAA          " ,
        "          AAAA          " ,
        "          AA          " };

#define SPRNUM 6 /* Number of sprites */
#define CWIDTH 38 /* Width of copyright message in characters */
#define CHEIGHT 6 /* Message height in rows */
#define SX (MAXX-(CWIDTH*8)) / 2 /* Starting X-coordinate */
#define SY (MAXY-(CHEIGHT*8)) / 2 /* Starting Y-coordinate */

struct { /* For sprite management */
    SPID *spidp; /* Pointer to sprite ID */
    int deltax[2], /* X-movement for pages 0 and 1 */
        deltax[2]; /* Y-movement for pages 0 and 1 */
} sprites[ SPRNUM ];

BYTE page, /* Current page */
out; /* Get flags for page collision */
int x, y, i, /* Loop counter */
dx, dy; /* Movement value */
char lc;
SPLOOK *starshipupp, *starshipdnp; /* Sprite pointer */

srand( *(int far *) 0x0040006c1 ); /* Initialize random numbers */

/*-- Create patterns for the different sprites -----*/

starshipupp = CompileSprite( StarShipUp, 20, 0, 'A', 1 );
starshipdnp = CompileSprite( StarShipDown, 20, 40, 'A', 1 );

/*-- Fill the first two graphic pages with characters -----*/
for ( page = 0; page < 2; ++ page )
{
    setpage( page );
    for ( lc = 0, y = 0; y < (MAXY+1)-8; y += 12 )
        for ( x = 0; x < (MAXX+1)-8; x += 8 )
            GrfxPrintf( x, y, lc % 255, 255, "%c", lc++ & 127 );

    /*-- Display copyright message -----*/

    Line( SX-1, SY-1, SX+CWIDTH*8, SY-1, 15 );
    Line( SX+CWIDTH*8, SY-1, SX+CWIDTH*8, SY+CHEIGHT*8, 15 );
    Line( SX+CWIDTH*8, SY+CHEIGHT*8, SX-1, SY+CHEIGHT*8, 15 );
    Line( SX-1, SY+CHEIGHT*8, SX-1, SY-1, 15 );
    GrfxPrintf( SX, SY, 15, 4,
        " );
    GrfxPrintf( SX, SY+8, 15, 4,
        " S3240C.C (c) 1992 by Michael Tischer " );
    GrfxPrintf( SX, SY+16, 15, 4,
        " );
    GrfxPrintf( SX, SY+24, 15, 4,
        " Sprite demo for 320x400 mode " );
    GrfxPrintf( SX, SY+32, 15, 4,
        " on VGA cards " );
    GrfxPrintf( SX, SY+40, 15, 4,
        " );
}

/*-- Create different sprites -----*/
for ( i = 0; i < SPRNUM ; ++ i)
{
    sprites[ i ].spidp = CreateSprite( starshipupp );
    do /* Select movement value for sprites */
    {
        dx = 0;
        dy = random(8) - 4;
    }
    while ( dx==0 && dy==0 );
}

```

```

sprites[ i ].deltax[0] = sprites[ i ].deltax[1] = dx * 2;
sprites[ i ].deltay[0] = sprites[ i ].deltay[1] = dy * 2;

x = ( 320 / SPRNUM * i ) + (320 / SPRNUM - 40) / 2 ;
y = random( (MAXY+1) - 40 );
SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
}

/*-- Move sprites and bounce them off the page borders -----*/

page = 1;                                /* Start with page 1 */
while ( !kbhit() )                       /* Press a key to end the loop */
{
    showpage( (BYTE) (1 - page) );        /* Display other page */

    for ( i = 0; i < SPRNUM; ++ i )       /* Execute sprites */
    {
        /* Move sprite and check for page collision */
        out = MoveSprite( sprites[i].spidp, page, sprites[i].deltax[page],
                           sprites[i].deltay[page] );
        if ( out & OUT_TOP || out & OUT_BOTTOM ) /* Top/bottom */
            /* collision? */
            {
                /* Yes --> Change direction of movement and sprite graphic */
                sprites[i].deltay[page] = 0 - sprites[i].deltay[page];
                sprites[i].spidp->splookp = ( out & OUT_TOP ) ? starshipdnp
                                                    : starshipupp;
            }
        if ( out & OUT_LEFT || out & OUT_RIGHT )
            sprites[i].deltax[page] = 0 - sprites[i].deltax[page];
    }
    page = (page+1) & 1;                  /* Toggle between 1 and 0 */
}

/*****
***                               M A I N   P R O G R A M                               ***
*****/

void main( void )
{
    union REGS regs;

    if ( IsVga() )                      /* VGA card installed? */
    {
        init320400();                  /* Yes --> Go ahead */
        Demo();                        /* Initialize graphic mode */
        getch();                       /* Wait for a key */
        regs.x.ax = 0x0003;            /* Shift into text mode */
        int86( 0x10, &regs, &regs );
    }
    else
        printf( "S3240C.C - (c) 1992 by Michael Tischer\n\n "
                "This program requires a VGA card\n\n" );
}

```