

```

/*****
*
*           S 6 4 3 5 C . C
*
**-----**
*   Task           : Demonstrates sprites in 640x350 EGA and VGA
*                   : graphic modes, using 16 colors and two screen
*                   : pages. This program requires assembler routines
*                   : from modules V16COLCA.ASM and S6435CA.ASM.
**-----**
*   Author          : Michael Tischer
*   Developed on    : 12/05/90
*   Last update     : 03/02/92
**-----**
*   Memory model    : SMALL
**-----**
*   (MICROSOFT C)
*   Compilation     : CL /AS /c W0 s6435c.c
*                   : LINK s6435c s6435ca v16colca;
**-----**
*   (BORLAND TURBO C)
*   Compilation     : Create a project file containing the following:
*                   : s6435c.c
*                   : v16colca.asm
*                   : s6435ca.asm
**-----**
*   Call            : s6435c
*****/

```

```

#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 {
    BYTE widthbytes,          /* Range width in bytes */
        numrows;             /* Number of rows */
    int  pixblen;             /* Length of pixel buffer */
    void *pixbptr;           /* Pointer to pixel buffer */
} PIXBUF;
typedef PIXBUF *PIXPTR;      /* Pointer to a pixel buffer */

typedef struct {
    BYTE  twidth,             /* Sprite design */
        theight;            /* Total width */
    void  *bmskp[8];         /* Height in pixel rows */
    PIXPTR pixmp[8];         /* Ptr to bit mask for AND */
} SPLLOOK;                  /* Pointer to pixel definition */

typedef struct {
    /* Sprite descriptor (ID) */

```

```

        SPLOOK *splookp;                /* Pointer to design */
        int     x[2], y[2];             /* Coordinates in pages 0 and 1 */
        PIXPTR  hgp[2];                 /* Pointer to background buffer */
    } SPID;

typedef struct {
        BYTE *fieldptr, /* Pointer to buffer with bit field */
        *curp[2], /* Pointer to currently processed byte */
        curbit, /* Currently processed bit in cur. byte */
        curbyte; /* Current byte value */
    }
    BITFIELD;
typedef BITFIELD *BFPTR;                /* Pointer to a bit field */

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

extern void init640350( void );
extern void setpix( int x, int y, unsigned char pcolor);
extern BYTE getpix( int x, int y );
extern void setpage( int page );
extern void showpage( int page );
extern void far * getfontptr( void );

extern void copybuf2video( BYTE *bufptr, BYTE page,
                           int tox, int toy, BYTE rwidth,
                           BYTE rheight );
extern void copyvideo2buf( BYTE *bufptr, BYTE page,
                           int fromx, int fromy, BYTE rwidth,
                           BYTE rheight );
extern void mergeandcopybuf2video( void * spribufptr, void * hgbufptr,
                                   void * andbufptr, BYTE page,
                                   int tox, int toy,

```

```

                                BYTE rwidth, BYTE rheight );

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

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

#define MAXX 639                /* Maximum X- and Y-coordinates */
#define MAXY 349

#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 EGA        0                /* Card types */
#define VGA        1
#define NEITHERNOR 2

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

/*****
 * IsEgaVga : Determines whether EGA or VGA card is installed.
 *-----*
 * Input   : None
 * Output  : EGA, VGA or NEITHERNOR
 *****/

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

```

```

Regs.x.ax = 0x1a00;          /* Function 1AH applies to VGA only */
int86( 0x10, &Regs, &Regs );
if ( Regs.h.al == 0x1a )      /* Function available? */
    return VGA;
else
{
    Regs.h.ah = 0x12;          /* Call function 12H, */
    Regs.h.bl = 0x10;          /* sub-function 10H */
    int86(0x10, &Regs, &Regs ); /* Call video BIOS */
    return (BYTE) (( Regs.h.bl != 0x10 ) ? EGA : NEITHERNOR);
}
}

/*****
*   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 section of 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 Y-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 )
        {

```



```

fptr = getfontptr();                                /* No --> Get pointer */

/*- Create 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 --> Set every pixel */
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 in the graphic screen.
* Input      : X, Y      = Starting coordinates (0 - ...)
*              fg        = Foreground color
*              bk        = Background color (255 = transparent)
*              STRING    = String with formatting information
*              ...       = arguments are 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 parameters */
vsprintf( stngbuf, string, parameter ); /* Format */
for ( cp = stngbuf; *cp; ++cp, x+= 8 ) /* Display formatted */
    PrintChar( *cp, x, y, fg, bk );      /* string using PrintChar */
}

/*****
*   GetVideo: Gets the contents of a rectangular range from video RAM
*               and puts them in a buffer.
**-----**
*   Input      : PAGE      = Screen page (0 or 1)
*                X1, Y1    = Starting coordinates
*                WRANGE    = Width of the rectangular range in pixels
*                HRANGE    = Height of rectangular range in pixels
*                BUFPTR    = Pointer to pixel buffer, in which the inform-
*                           ation is to be placed
*   Output     : Pointer to created pixel buffer with the contents of the
*                specified range
*   Info       : If the BUFPTR parameter passes the ALLOCBUF value, a new
*                pixel buffer is allocated using the heap, then returned.
*                This buffer can be specified again for a new call,
*                unless the previous contents are still required and the
*                size of the rectangular area remains unchanged compared
*                to the preceding call.
*                The specified area must begin at an X-coordinate that
*                can be divided by eight and extend over a multiple of
*                eight pixels.
*****/

```

```

PIXPTR GetVideo( BYTE page, int x1, int y1, BYTE wrange, BYTE hrange,
                PIXPTR bufptr )
{
    if ( bufptr == ALLOCBUF )           /* No buffer passed during call? */
    {                                   /* No --> Create one */
        bufptr = malloc( sizeof( PIXBUF ) );      /* Create pixel buffer */
        bufptr->pixbptr = malloc( (wrange*hrange) / 2 ); /* Alloc. px.b. */
        bufptr->numrows = hrange;                /* Height of buffer in lines */
        bufptr->widthbytes = wrange / 8;          /* Width of a line in bytes */
        bufptr->pixblen = (wrange*hrange) / 2;    /* Total length of buffer */
    }

    copyvideo2buf( bufptr->pixbptr, page, x1, y1, wrange / 8, hrange );
    return bufptr;                          /* Return pointer and buffer to caller */
}

/*****
* PutVideo: Writes the contents of a rectangular area of the screen *
*           previously saved by GetVideo back to the video RAM.      *
**-----**
* Input   : BUFPTR = Pointer to pixel buffer returned during      *
*           previous call for GetVideo                             *
*           PAGE   = Screen page (0 or 1)                          *
*           X1, Y1 = Starting coordinates                          *
* Info    : This procedure does not delete the pixel buffer. The  *
*           FreePixBuf procedure must be called for this.         *
*           The specified X-coordinate must be a multiple of eight! *
*****/

void PutVideo( PIXPTR bufptr, BYTE page, int x1, int y1 )
{

```

```

copybuf2video( bufptr->pixbptr, page, xl, yl,
               bufptr->widthbytes, bufptr->numrows );
}

```

```

/*****
*   FreePixBuf: Clears a pixel buffer allocated by the heap when
*               GetVideo was called.
* *****/
* Input   : BUFPTR = Pointer to pixel buffer returned during
*           previous call for GetVideo
*****/

```

```

void FreePixBuf( PIXPTR bufptr )
{
    free( bufptr->pixbptr );
    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
* Info    : Sprite backgrounds comprise two adjacent areas the same
*           size as the sprites.
*****/

```

```

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

```



```

}

/*****
* BfAppendBit: Appends a bit to a bit field.
**-----**
* Input   : BFID = Pointer to the bit field descriptor, returned by
*           the call to BfInit()
*           BIT   = Values of bits to be appended (0 or 1)
* Output  : None
*****/

void BfAppendBit( BFPTR bfid, BYTE bit )
{
    bfid->curbyte |= bit;           /* Add bit at bit position 0 */
    if ( bfid->curbit == 7 )        /* Byte already full? */
    {                               /* Yes */
        *(bfid->curptr++) = bfid->curbyte; /* Place byte in buffer */
        bfid->curbyte = bfid->curbit = 0; /* Bit mask reverts to 0 */
    }
    else                            /* Byte still not full */
    {
        ++bfid->curbit;            /* Process another bit */
        bfid->curbyte <= 1;        /* Shift bit mask */
    }
}

/*****
* BfEnd : Ends bit field processing, clears descriptor without
*         clearing the bit field.
**-----**
* Input   : BFID = Pointer to the bit field descriptor returned
*           after a call to BfInit()
*****/

```

```

*   Output   : Pointer to the bit field whose buffer can be released by *
*               FREE().
*
*****/

```

```

void *BfEnd( BFPTR bfid )
{
    void *retptr;                                /* Pointer to bit field */

    if ( bfid->curbit )                          /* Last byte still not full? */
        *bfid->curptr = bfid->curbyte << (7 - bfid->curbit ); /*No --> Close*/

    retptr = bfid->fieldptr;                      /* Store pointer to bit field */
    free( bfid );                                /* Free descriptor */

    return retptr;                              /* Return pointer to bit field */
}

```

```

/*****
*   CompileSprite: Creates a sprite's pixel and bit patterns, based on *
*                   the sprite's definition at runtime.
*
**-----**
*   Input      : BUFP      = Pointer to array contains string pointers
*                   controlling sprite's pattern
*
*               SHEIGHT = Sprite height and number of strings needed
*
*   Info       : In passed sprite pattern, a space represents a background*
*                   pixel, the A represents color code 0, B represents 1,
*                   1, C represents 2, etc.
*
*****/

```

```

SPLOOK *CompileSprite( char **bufp, BYTE sheight )
{
    BYTE    stwidth,                                /* String width */

```

```

        spwidth,                                /* Sprite width */
        c,                                       /* Get character from c sprite array */
        i, k, l, y;                             /* Loop variables */
SPLOOK *splookp;                               /* Pointer to created sprite structure */
PIXPTR tpix;                                  /* Get sprite background temporarily */
BFPTR bfptr;                                   /* Pointer to bit field descriptor */

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

splookp = (SPLOOK *) malloc( sizeof(SPLOOK) );
stwidth = (BYTE) strlen( *bufp ); /* String length and logo width */
spwidth = ( ( stwidth + 7 + 7 ) / 8 ) * 8; /* Total width */
splookp->twidth = spwidth;
splookp->theight = sheight;

setpage( 1 );                                /* Draw sprites in page 1 */
showpage( 0 );                               /* but show page 0 */

tpix = GetVideo( 1, 0, 0, spwidth, sheight, ALLOCBUF ); /* Store bkg */

/*-- Draw and code sprite eight times -----*/

for ( l = 0; l < 8; ++l )
{
    /* Fill background with black pixels */
    for ( y = 0; y < sheight; ++y )
        Line( 0, y, spwidth-1, y, 0 );

    bfptr = BfInit( spwidth*sheight ); /* Alloc. mem. for AND buffer */

    for ( i = 0; i < sheight ; ++i )
    {
        /* Execute lines */
        for ( y = l; y; --y )
            /* Create AND bits for left border */

```



```

        BfAppendBit( bfptr, 1 );

for ( k = 0; k < stwidth; ++k )                /* Execute columns */
{
    if ( ( c = (*(bufp+i)+k) ) == 32 )          /* Background pixel? */
    {                                           /* Yes --> Color code 0 */
        setpix( k+1, i, 0 );
        BfAppendBit( bfptr, 1 );              /* Background pixel remaining? */
    }
    else                                        /* No --> Color code as specified */
    {
        setpix( k+1, i, c-64 );
        BfAppendBit( bfptr, 0 );              /* Mask background pixels */
    }
}

for ( y = spwidth-stwidth-1; y ; --y )        /* Append AND bits for */
    BfAppendBit( bfptr, 1 );                  /* right border */
}
splookp->bmskp[ 1 ] = BfEnd( bfptr );

/*-- Get sprite pixel pattern from video RAM -----*/
splookp->pixmp[ 1 ] = GetVideo( 1, 0, 0, spwidth, sheight, ALLOCBUF );
}                                           /* Design first of eight sprites */

PutVideo( tpix, 1, 0, 0 );                  /* Restore sprite background in */
FreePixBuf( tpix );                          /* page 1 and clear 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 )
{
    int x;                                /* X-coordinate of sprite */

    x = spidp->x[sprpage];
    mergeandcopybuf2video( spidp->splookp->pixmp[x % 8]->pixbptr,
                           spidp->hgptr[sprpage]->pixbptr,
                           spidp->splookp->bmskp[x % 8],
                           sprpage,
                           x & (~7),
                           spidp->y[sprpage],
                           spidp->splookp->twidht / 8,
                           spidp->splookp->theight );
}

```

```

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

```

```

void GetSpriteBg( register SPID *spidp, BYTE sprpage )
{
    GetVideo( sprpage, spidp->x[sprpage] & (~7), 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] & (~7), spidp->y[sprpage] );
}

/*****
*   MoveSprite: Copies sprite in 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 */

/*-- 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->twidht )
    {
        newx = (2*(MAXX+1))-newx-2*(spidp->splookp->twidht);
        out = OUT_RIGHT;
    }
    else
        out = OUT_NO;

/*-- 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 border? */
    {
        /* 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] )
{
    RestoreSpriteBg( spidp, sprpage ); /* If there's a new position */
    spidp->x[sprpage] = newx;           /* then reset background and */
    spidp->y[sprpage] = newy;           /* store new coordinates */
    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 1 and 0  */
}

```

```

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

```

```

void RemoveSprite( SPID *spidp )
{
    RestoreSpriteBg( spidp, 0 );    /* Restore sprite background */
    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          " ,
"          GBBCCBBBG        " ,
"          GBBBBBBBBBBG      " ,
"          GBBBBBBBBBBG      " ,
" G          GBBBBBBBBBBBG    G " ,
"GCG      GGBBBDBBB  BBBDBBBGG  GCG" ,
"GCBGGGBBBBBDBB  BBDBBBBGGGBCG" ,
"GCBBBBBBBBBBDB  BDBBBBBBBBBBCG" ,
"BBBBBBBBBBBBBDB BB BDBBBBBBBBBBB" ,
"GGBBBBBBBDBBBBBBBBBBDBBBBBBCG" ,
"  GGCCBBBDDDDDDDDDDDDDBBCCG " ,
"    GGBBDDDDGGGGGDDDDDBBG    " ,
"      GDDDDGGG    GGGDDDDG    " ,
"          DDDD          DDDD    " } ;

```

```

static char *StarShipDown[20] =
{

```

```

"          DDDD          DDDD    " ,
"      GDDDDGGG    GGGDDDDG    " ,
"    GGBBDDDDGGGGGDDDDDBBG    " ,
"  GGCCBBBDDDDDDDDDDDDDBBCCG " ,
"GGBBBBBBBDBBBBBBBBBBDBBBBBBCG" ,
"BBBBBBBBBBBBBDB BB BDBBBBBBBBBBB" ,
"GCBBBBBBBBBBDB  BDBBBBBBBBBBCG" ,
"GCBGGGBBBBBDBB  BBDBBBBGGGBCG" ,
"GCG  GGBBBDBBB  BBBDBBBGG  GCG" ,
"GCG      GGBBBBBBBBBBDGG      GCG" ,
" G          GBBBBBBBBBBBG    G " ,
"          GBBBBBBBBBBG      " ,
"          GBBBBBBBBBBG      " ,

```

```

"          GBBBCCBBBG          " ,
"          GBBCCBBG           " ,
"          GBBBGG             " ,
"          AA                  " ,
"          AAAA                " ,
"          AAAA                " ,
"          AA                   " };

```

```

#define SPRNUM 6                                /* Number of sprites */
#define CWIDTH 42                             /* Width of copyright message in characters */
#define CHEIGHT 6                             /* Height in rows */
#define SX      (MAXX-(CWIDTH*8)) / 2         /* Starting X-coordinate */
#define SY      (MAXY-(CHEIGHT*8)) / 2         /* Starting Y-coordinate */

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

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;           /* Pointer to sprites */

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

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

starshipupp = CompileSprite( StarShipUp,  20 );
starshipdnp = CompileSprite( StarShipDown, 20 );

```



```

/*-- Fill the first two graphic pages with characters -----*/
for ( page = 0; page < 2; ++ page )
{
    setpage( page );
    showpage( 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 & 15, 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+HEIGHT*8, 15 );
    Line( SX+CWIDTH*8, SY+HEIGHT*8, SX-1, SY+HEIGHT*8, 15 );
    Line( SX-1, SY+HEIGHT*8, SX-1, SY-1, 15 );
    GrfxPrintf( SX, SY, 15, 4,
                " " );
    GrfxPrintf( SX, SY+8, 15, 4,
                " S6435C.C - (c) 1992 by Michael Tischer " );
    GrfxPrintf( SX, SY+16, 15, 4,
                " " );
    GrfxPrintf( SX, SY+24, 15, 4,
                " Sprite demo for 640x350 mode " );
    GrfxPrintf( SX, SY+32, 15, 4,
                " on EGA and VGA cards " );
    GrfxPrintf( SX, SY+40, 15, 4,
                " " );
}

/*-- Create sprites -----*/

```

```

for ( i = 0; i < SPRNUM ; ++ i)
{
    sprites[ i ].spidp = CreateSprite( starshipupp );
    do
        /* Select movement values 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 = ( MAXX / SPRNUM * i ) + (MAXX / 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;
while ( !kbhit() )
{
    /* Start with page 1 */
    /* 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 )
        {
            /*T/B collision?*/
            /* Yes --> Change direction 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 ( IsEgaVga() != NEITHERNOR )                /* EGA or VGA card installed? */
    {
                                                /* Yes --> Go ahead */
        init640350();
                                                /* Initialize graphic mode */
        Demo();
        getch();
                                                /* Wait for a key */
        regs.x.ax = 0x0003;
                                                /* Shift to text mode */
        int86( 0x10, &regs, &regs );
    }
    else
        printf( "S6435C.C - (c) 1992 by Michael Tischer\n\n"
                "This program requires an EGA or a VGA card\n\n" );
}

```

$\tilde{Y} \bullet$