

```

{*****
*
*               S 3 2 2 0 P . P A S
*
**-----**
* Task          : Demonstrates sprites in 320x200 VGA graphic
*                mode, using 256 colors and four screen pages.
*                This program requires the assembly language
*                modules V3220PA.ASM and S3220PA.ASM.
*
**-----**
* Author        : Michael Tischer
* Developed on   : 09/12/90
* Last update    : 02/12/92
*****}

```

program S3220P;

uses dos, crt;

```

{-- External references to the assembler routines -----}

```

```

{$L v3220pa}                                { Link assembler module }

```

```

procedure init320200; external;
procedure setpix( x, y : integer; pcolor : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

```

```

{$L s3220pa}                                { Link assembler module }

```

```

procedure blockmove( frompage : byte; fromx, fromy : integer;
                    topage : byte; tox, toy : integer;
                    pwidth, pheight: byte; bmskp : pointer ); external;

```

```

{-- Constants -----}

const MAXX = 319;           { Maximum X- and Y-coordinates }
    MAXY = 199;

    OUT_LEFT   = 1;   { For collision documentation in SpriteMove() }
    OUT_TOP    = 2;
    OUT_RIGHT  = 4;
    OUT_BOTTOM = 8;
    OUT_NO     = 0;           { None }

{-- Type declarations -----}

type SPLOOK = record
    twidth,           { Sprite design }
    theight,          { Total width }
    ppage,            { Height in pixel lines }
    msklen : byte;    { Placed in page ... }
    bmskp  : pointer; { Entry length }
    pxlin  : integer; { Pointer to bit mask }
    end;            { Pixel lines for sprite }
SPLP = ^SPLOOK;    { in its page }
                { Pointer to sprite design }

SPID = record
    bkgpage : byte;    { Sprite descriptor }
    x, y : array [0..1] of integer; { Background page }
    bkx, bky : integer; { Coordinates: pp. 0 & 1 }
    splookp : SLPK;    { Background buffer }
    end;            { Pointer to sprite design }
SPIP = ^SPID;      { Pointer to sprite descriptor }

```

```

    BYTEAR = array[0..10000] of byte;           { Addressing      }
    BARPTR = ^BYTEAR;                           { different buffers }

    PTRREC = record                               { For pointer or LONGINTS analysis }
        ofs,
        seg : word;
    end;

{ *****
*   IsVga : Determines whether a VGA card is installed.
*   -----**
*   Input   : None
*   Output  : TRUE or FALSE
*   *****}

function IsVga : boolean;

var Regs : Registers;           { Processor registers for interrupt call }

begin
    Regs.AX := $1a00;            { Function 1AH applies to VGA only }
    Intr( $10, Regs );
    IsVga := ( Regs.AL = $1a );
end;

{ *****
*   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.                                             *
*****}

```

```

procedure PrintChar( thechar : char; x, y : integer; fg, bk : byte );

type FDEF = array[0..255,0..7] of byte;           { Font array }
      TPTR = ^FDEF;                               { Pointer to font }

var  Regs   : Registers;                          { Registers for interrupt call }
      ch    : char;                               { Individual pixels in character }
      i, k,                               { Loop counter }
      BMask : byte;                               { Bit mask for character design }

const fptr : TPTR = NIL;                          { Pointer to font in ROM }

begin
  if fptr = NIL then                               { Pointer to font already set? }
    begin                                           { No }
      Regs.AH := $11;                               { Call video BIOS function 11H, }
      Regs.AL := $30;                               { sub-function 30H }
      Regs.BH := 3;                                 { Get pointer to 8x8 font }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );               { Set pointers }
    end;

  if ( bk = 255 ) then                               { Drawing transparent characters? }
    for i := 0 to 7 do                             { Yes --> Set foreground pixels only }
      begin
        BMask := fptr^[ord(thechar),i]; { Get bit pattern for one line }
        for k := 0 to 7 do
          begin

```

```

        if ( BMask and 128 <> 0 ) then                { Pixel set? }
            setpix( x+k, y+i, fg );                    { Yes }
            BMask := BMask shl 1;
        end;
    end
else                { No --> Consider background as well }
    for i := 0 to 7 do                { Execute lines }
        begin
            BMask := fptr^[ord(thechar),i];{ Get bit pattern for one line }
            for k := 0 to 7 do
                begin
                    if ( BMask and 128 <> 0 ) then        { Foreground? }
                        setpix( x+k, y+i, fg )            { Yes }
                    else
                        setpix( x+k, y+i, bk );            { No --> Background }
                        BMask := BMask shl 1;
                    end;
                end;
            end;
        end;
    end;
end;

```

```

{*****}
*   PrintString: Writes a string to the screen in graphics mode.   *
*-----*
*   Input      :   X, Y      = X- and Y-coordinates of upper left-corner *
*                :   FG      = Foreground color                        *
*                :   BK      = Background color                        *
*                :   TSTR     = String to be displayed                 *
*   Info       :   The characters are designed around an 8x8 matrix, based *
*                :   on the 8x8 ROM font.                             *
{*****}

```

```

procedure PrintString( x, y : integer; fg, bk : byte; tstr : string );

var i : integer;                                { Loop counter }

begin
  for i := 1 to length( tstr ) do                { Execute string }
  begin
    PrintChar( tstr[i], x, y, fg, bk );           { and display it }
    inc( x, 8 );                                   { Increment output position }
  end;
end;

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

procedure Line( x1, y1, x2, y2 : integer; lpcol : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procedure for swapping two integer variables -----}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

```

```

begin
    dummy := i2;
    i2     := i1;
    i1     := dummy;
end;

{-- Main procedure -----}

begin
    if ( abs(x2-x1) < abs(y2-y1) ) then          { X- or Y-axis overflow? }
        begin                                    { Check Y-axes }
            if ( y1 > y2 ) then                  { y1 > y2? }
                begin
                    SwapInt( x1, x2 );           { Yes --> Swap X1 with X2 }
                    SwapInt( y1, y2 );           { and Y1 with Y2 }
                end;
            if ( x2 > x1 ) then xincr := 1        { Set X-axis increment }
                else xincr := -1;

            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 to y2 do                  { Execute line on Y-axes }
                begin

```

```

        if ( d >= 0 ) then
            begin
                inc( x, xincr );
                inc( d, aincr );
            end
        else
            inc( d, bincr );
            setpix( x, y, lpcol );
        end;
    end
else
    begin
        { Check X-axes }
        if ( x1 > x2 ) then
            { x1 > x2? }
            begin
                SwapInt( x1, x2 );
                SwapInt( y1, y2 );
                { Yes --> Swap X1 with X2 }
                { and Y1 with Y2 }
            end;

            if ( y2 > y1 ) then yincr := 1
                else yincr := -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 );
            for x:=x1+1 to x2 do
                begin
                    { Set first pixel }
                    { Execute line on X-axes }

```



```

        if ( d >= 0 ) then
            begin
                inc( y, yincr );
                inc( d, aincr );
            end
        else
            inc( d, bincr );
            setpix( x, y, lpcol );
        end;
    end;
end;

```

```

{*****
* GrfxPrint: 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      *
*****}

```

```

procedure GrfxPrint( x, y : integer; fg, bk : byte; strt : string );

var i : integer;                                { Loop counter }

begin
    for i:=1 to length( strt ) do
        begin
            printchar( strt[i], x, y, fg, bk );    { Display using PrintChar }
            inc( x, 8 );                            { Move X to next character position }
        end;
    end;
end;

```

```

{*****
*   CreateSprite: Creates a sprite based on a user-defined      *
*                   pixel pattern.                               *
**-----**
*   Input   : SPLOOKP = Pointer to data structure from CompileSprite *
*             BKGPAGE = Screen page in which sprite background should *
*                   be stored                                         *
*             BKX,    = bkgpage coordinates at which sprite background *
*             BKY      is stored                                       *
*   Output  : Pointer to created sprite structure                  *
*   Info   : The sprite background requires two areas the same size *
*           as the corresponding sprite.                             *
*****}

```

```

function CreateSprite( splookp : SPLP; bkgpage : byte;
                      bkx, bky : integer           ) : SPIP;

```

```

var spidp : SPIP;           { Pointer to created sprite structure }

```

```

begin
  new( spidp );              { Allocate memory for sprite descriptor }
  spidp^.splookp := splookp;  { Pass data to the }
  spidp^.bkgpage := bkgpage;  { sprite structure }
  spidp^.bkx := bkx;
  spidp^.bky := bky;

  CreateSprite := spidp;     { Return pointer to the sprite structure }
end;

```

```

{*****
*   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) *
*               GPAGE   = Graphic page for sprite design          *
*               Y       = Pixel lines needed for sprite           *
*               FB      = ASCII characters for the smallest color  *
*               FGCOLOR = First color code for FB                  *
*   Info     : Sprite structure of pixel lines starts at left margin. *
*****}

```

```

function CompileSprite( var buf; sheight, gpage : byte;
                       y : integer; fb : char; fgcolor : byte ) : SPLP;

```

```

type BYPTR = ^byte;                                { Pointer to a byte }

```

```

var  swidth,                               { String width }
    c,                                     { Get character from c sprite array }
    cvcolor,                             { Converted color of a pixel }
    i, k, l,                             { Loop variables }
    pixc,                                { Pixel counter for creating the bit mask }
    pixm      : byte;                    { Pixel mask }
    spacing,   { Spacing from start of sprite to start of sprite }
    lx, ly    : integer;                 { Floating coordinates }
    splookp   : SPLP;                   { Pointer to created sprite structure }
    lspb      : BYPTR;                   { Floating pointer in sprite buffer }
    bptr      : barptr;                  { Addresses buffer with graphic }

```

```

begin
  {-- Create SpriteLook structure and fill with data -----}

```

```

new( splookp );
bptr := @buf;                                { Set pointer to logo buffer }
swidth := bptr^[0];    { Get string length and determine logo width }
spacing := ( ( swidth + 3 + 3 ) div 4 ) * 4;
splookp^.twidth := spacing;
splookp^.msklen := (spacing*sheight+7) div 8;
getmem( splookp^.bmskp, splookp^.msklen * 4 );
splookp^.theight := sheight;
splookp^.pxlin   := y;
splookp^.ppage   := gpage;

{-- Fill sprite background in home page with -----}
{-- codes for transparent character background -----}

setpage( gpage );                                { Set page for drawing }
lx := 4 * spacing - 1;
for ly:=y+sheight-1 downto y do
    Line( 0, ly, lx, ly, 255 );

{-- Draw four matching sprites in the home page -----}

lx := 0;                                { Start at left border of line }
for l := 1 to 4 do                        { Draw sprite four times }
    begin
        for i := 0 to sheight-1 do        { Execute rows }
            for k := 0 to swidth-1 do      { Execute columns }
                begin
                    c := bptr^[i*(swidth+1)+k+1];    { Get color }
                    if ( c = 32 ) then                { Background pixel? }
                        setpix( lx+k, y+i, 255 )    { Yes --> Set color code 255 }
                    else                               { No --> Set color code as given }
                        setpix( lx+k, y+i, fgcolor+(c-ord(fb)) );
                end
            end
        end
    end

```



```

begin
    lspb^ := pixm shr 4;          { No --> Move high nibble to }
    pixc := 0;                    { low nibble and store them }
    pixm := 0;                    { Set pixel counter and mask to 0 }
end;

inc( lx, spacing );              { LX to start of next sprite }
end;

CompileSprite := splookp;        { Return pointer to sprite buffer }
end;

{*****
*   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) *
*****}

procedure PrintSprite( spidp : SPIP; sprpage : byte );

var twidth : byte;                { Total width of sprite }
    x      : integer;            { X-coordinate of sprite in its page }
    splookp : SPLP;              { Pointer to sprite's appearance }

begin
    splookp := spidp^.splookp;
    twidth  := splookp^.twidth;
    x       := spidp^.x[sprpage];
    blockmove( splookp^.ppage, twidth * (x mod 4), splookp^.pxlin, sprpage,
        x and not(3), spidp^.y[sprpage], twidth, splookp^.theight,
        @BARPTR(splookp^.bmskp)^[(x mod 4) * splookp^.msklen] );

```



```

var splookp : SPLP;          { Pointer to sprite graphic }

begin
  splookp := spidp^.splookp;
  blockmove( spidp^.bkpage, spidp^.bkx + ( splookp^.twidth * sprpage ),
            spidp^.bky, sprpage, spidp^.x[sprpage], spidp^.y[sprpage],
            splookp^.twidth, splookp^.theight, NIL );
end;

{ *****
*   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)                  *
*   *****}

function MoveSprite( spidp : SPIP; sprpage : byte;
                    deltax, deltax : integer      ) : byte;

var newx, newy : integer;          { New sprite coordinates }
    out        : byte;            { Display collision with border }

begin
  {-- Move X-coordinates and test for border collision -----}

  newx := spidp^.x[sprpage] + deltax;
  if ( newx < 0 ) then
    begin

```



```

    newx := 0 - deltax - spidp^.x[sprpage];
    out := OUT_LEFT;
end
else
    if ( newx > MAXX - spidp^.splookp^.twidth ) then
        begin
            newx := (2*(MAXX+1))-newx-2*(spidp^.splookp^.twidth);
            out := OUT_RIGHT;
        end
    else
        out := OUT_NO;
    end

{-- Move Y-coordinates and test for border collision -----}

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

{-- Set new position only if different from old position -----}

if ( newx <> spidp^.x[sprpage] ) or ( newy <> spidp^.y[sprpage] ) then
    begin                                     { If there's a new position }
        RestoreSpriteBg( spidp, sprpage );   { then reset background and }
    end
end

```

```

    spidp^.x[sprpage] := newx;           { store new coordinates      }
    spidp^.y[sprpage] := newy;
    GetSpriteBg( spidp, sprpage );       { Get new background      }
    PrintSprite( spidp, sprpage );       { Draw sprite in specified page }
end;

```

```

MoveSprite := out;
end;

```

```

{*****}
*  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.                  *
*****}

```

```

procedure SetSprite( spidp : SPIP; x0, y0, x1, y1 : integer );

```

```

begin
    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 }
end;

```



```

'          GBBBBBBBBBBG          ' ,
'          GBBBBBBBBBBG          ' ,
'  G          GBBBBBBBBBBBBG          G  ' ,
' GCG          GGDBBBBBBBBBBDDGG          GCG ' ,
' GCG          GGBBBD BBB          BBBDBBBGG          GCG ' ,
' GCBGGGBBBBBDBB          BBDBBBBBGGGBCG ' ,
' GCBBBBBBBBBBDB          BDBBBBBBBBBBBG ' ,
' BBBBBBBBBBBBBBDB BB          BDBBBBBBBBBBBB ' ,
' GGCBBBBBBBDBBBBBBBBBBBDBBBBBBBBCG ' ,
'          GGCCBBBDDDDDDDDDDDDDBBBCCG ' ,
'          GGBBDDDDGGGGGGDDDDDBBG ' ,
'          GDDDDGGG          GGGDDDDG ' ,
'          DDDD          DDDD          ' );

```

```

const StarShipDown :array [1..20] of string[32] =
(
'          DDDD          DDDD          ' ,
'          GDDDDGGG          GGGDDDDG          ' ,
'          GGBBDDDDGGGGGGDDDDDBBG          ' ,
'          GGCCBBBDDDDDDDDDDDDDBBBCCG          ' ,
' GGCBBBBBBBDBBBBBBBBBBBDBBBBBBBBCG ' ,
' BBBBBBBBBBBBBBDB BB          BDBBBBBBBBBBBB ' ,
' GCBBBBBBBBBBDB          BDBBBBBBBBBBBG ' ,
' GCBGGGBBBBBDBB          BBDBBBBBGGGBCG ' ,
' GCG          GGBBBD BBB          BBBDBBBGG          GCG ' ,
' GCG          GGDBBBBBBBBBBDDGG          GCG ' ,
'  G          GBBBBBBBBBBBBG          G  ' ,
'          GBBBBBBBBBBG          ' ,
'          GBBBBBBBBBBG          ' ,
'          GBBCCBBG          ' ,
'          GBBCCBBG          ' ,
'          GGBGG          ' ,
'          AA          ' ,

```

```

      '          AAAA          ' ,
      '          AAAA          ' ,
      '          AA           ' );

```

```

SPRNUM = 6;                                { Number of sprites }
CWIDTH = 37;                             { Width of copyright message in characters }
HEIGHT = 6;                               { Message height in rows }
SX      = (MAXX-(CWIDTH*8)) div 2;        { Starting X-coordinate }
SY      = (MAXY-(HEIGHT*8)) div 2;        { Starting Y-coordinate }

```

```

type SPRITE = record                      { For sprite management }
    spidp : SPIP;                        { Pointer to sprite ID }
    deltax,                                { X-movement for pages 0 and 1 }
    delay : array [0..1] of integer;    { Y-movement }
end;

var sprites      : array [1..sprnum] of SPRITE;
    page,
    lc,                                { Current page }
    out          : byte;                { Character for screen design }
    x, y, i,
    dx, dy       : integer;            { Get flags for page collision }
    starshipupp,                                { Loop counter }
    starshipdnp : SPLP;                  { Movement value }
    ch           : char;                { Sprite pointer }
begin

```

```

    Randomize;                            { Initialize random number generator }

    {-- Fill the first two graphic pages with characters -----}

```

```

for page := 0 to 1 do
begin
  setpage( page );
  lc := 0;
  y := 0;
  while ( y < 200-8 ) do
    begin
      x := 0;
      while ( x < 320-8 ) do
        begin
          PrintChar( chr(lc and 127), x, y, lc mod 255, 0 );
          inc( lc );
          inc( x, 8 );
        end;
        inc( y, 12 );
      end;
    end;

  {-- 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 );
  PrintString( SX, SY, 15, 4,
    '
  );
  PrintString( SX, SY+8, 15, 4,
    ' *S3220P.PAS - (c) 1992 M. Tischer* ' );
  PrintString( SX, SY+16, 15, 4,
    '
  );
  PrintString( SX, SY+24, 15, 4,
    ' Sprite demo for 320x200 mode ' );
  PrintString( SX, SY+32, 15, 4,

```

```

                                '   on VGA cards           ' );
PrintString( SX, SY+40, 15, 4,
                                '   );

end;

{-- Create patterns for the different sprites -----}

starshipupp := CompileSprite( StarShipUp,   20, 2, 0, 'A', 1 );
starshipdnp := CompileSprite( StarShipDown, 20, 2, 40, 'A', 1 );

{-- Create different sprites -----}
for i := 1 to SPRNUM do
begin
    sprites[ i ].spidp := CreateSprite( starshipupp, 3, (i mod 3)*100,
                                         (i div 3) * 30 );
    repeat
        { Select movement values for sprites }
        dx := 0;
        dy := random(8) - 4;
    until ( dx <> 0 ) or ( dy <> 0 );

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

    x := ( (320 div SPRNUM) * (i-1) ) + ((320 div SPRNUM) - 40) div 2 ;
    y := random( 200 - 40 );
    SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
end;

{-- Move sprites and bounce them off the page borders -----}

```

```

page := 1;                                { Start with page 1 }
while ( not keypressed ) do                { Press a key to end the loop }
begin
    showpage( 1 - page );                  { Display other page }

    for i := 1 to sprnum do                { Execute sprites }
    begin
        { Move sprite and check for page collision }
        out := MoveSprite( sprites[i].spidp, page,
            sprites[i].deltax[page],
            sprites[i].deltay[page] );
        if ( ( out and OUT_TOP ) <> 0 ) or { Top/bottom collision? }
            ( ( out and OUT_BOTTOM ) <> 0 ) then
        begin
            { Yes --> Change direction of movement and sprite graphic }
            sprites[i].deltay[page] := -sprites[i].deltay[page];
            if ( ( out and OUT_TOP ) <> 0 ) then
                sprites[i].spidp^.splookp := starshipdnp
            else
                sprites[i].spidp^.splookp := starshipupp;
        end;
        if ( ( out and OUT_LEFT ) <> 0 ) or { Left/right collision? }
            ( ( out and OUT_RIGHT ) <> 0 ) then
            sprites[i].deltax[page] := -sprites[i].deltax[page];
        end;
        page := (page+1) and 1;             { Toggle between 1 and 0 }
    end;
ch := readkey;                             { Remove key from keyboard buffer }
end;

{ *****
*                               M A I N   P R O G R A M                               *
***** }

```



```

begin
  if ( IsVga ) then
    begin
      init320200;
      Demo;
      Textmode( CO80 );
    end
  else
    writeln( 'S3220P.PAS - (c) 1990 by Michael Tischer'#13#10#10 +
      'This program requires a VGA card'#13#10 );
  end.

```