

```

{
*****
*                               L O G O P . P A S
*                               *****
**-----**
* Task       : Demonstrates custom character definition for EGA
*             and VGA cards, for use as a logo design.
*             *****
**-----**
* Author      : Michael Tischer
* Developed on : 08/05/90
* Last update  : 02/21/92
*****}

Program Logop;

uses DOS, CRT;

{-- Constants -----}

const EGAVGA_SEQUENCER = $3C4;      { Sequencer address/data port }
      EGAVGA_MONCTR    = $3D4;      { Monitor controller }
      EGAVGA_GRAPHCTR  = $3CE; { Graphics controller address/data port }
      EV_STATC         = $3DA;      { EGA/VGA color status register }
      EV_STATM         = $3BA;      { EGA/VGA mono status register }
      EV_ATTR          = $3C0;      { EGA/VGA color attribute controller }

{-- Type declarations -----}

type CRDTYPE = ( EGA, VGA, NEITHERNOR );

procedure CLI; inline( $FA );      { Disable interrupts }
procedure STI; inline( $FB );      { Enable interrupts }

{*****}
* SetCharWidth: Sets VGA character width to 8 or 9 pixels.
*             *****
* Input      : HWIDTH = Character width (8 or 9)
*             *****}

procedure SetCharWidth( hwidth : byte );

var Regs : Registers;      { Processor registers for interrupt call }
    x    : byte;           { Value for misc. output reg. }

begin
  if ( hwidth = 8 ) then Regs.BX := $0001      { BH = horiz. direction }
    else Regs.BX := $0800;      { BL = seq. reg. value }

  x := port[ $3CC ] and not(4+8);      { Toggle horizontal }
  if ( hwidth = 9 ) then                { resolution from }
    x := x or 4;                        { 720 to 640 pixels }
  port[ $3C2 ] := x;

  CLI;      { Toggle sequencer from 8 to 9 pixels }
  portw[ EGAVGA_SEQUENCER ] := $0100;
  portw[ EGAVGA_SEQUENCER ] := $01 + Regs.BL shl 8;
  portw[ EGAVGA_SEQUENCER ] := $0300;
  STI;

  Regs.AX := $1000;      { Change screen configuration }
  Regs.BL := $13;
  intr( $10, Regs );
end;

{*****}
* IsEgaVga : Determines whether an EGA or a VGA card is installed.
*             *****
* Input      : None
* Output     : EGA, VGA or NEITHERNOR
*             *****}

function IsEgaVga : CRDTYPE;

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

begin
  Regs.AX := $1a00;      { Function 1AH applies only to VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then      { Is the function available? }
    IsEgaVga := VGA
  else
    begin
      Regs.ah := $12;      { Call function 12H, }
      Regs.bl := $10;      { sub-function 10H }
      intr($10, Regs);      { Call video BIOS }
      if ( Regs.bl <> $10 ) then IsEgaVga := EGA
        else IsEgaVga := NEITHERNOR;
    end;
end;

```

```

end;
{*****}
* BuildLogo : Draws a logo on the screen using custom characters *
* based on existing ASCII characters. *
*-----*
* Input      : COLUMN = Starting column of logo (1-80) *
*             LGROW   = Starting row of logo (1-25) *
*             DEPTH    = Number of logo scan lines *
*             OPCOL    = Logo output color *
*             BUF      = String array containing character patterns for *
*                       logo *
* Info       : - The Test procedure demonstrates how the logo buffer *
*               works. *
*             - The logo is displayed centered in a block *
*               of characters. *
{*****}

procedure BuildLogo( column, lgrow, depth, opcol : byte; var buf );

type BYTEAR = array[0..10000] of byte;           { Byte array for }
      BARPTR = ^BYTEAR;                         { logo buffer   }

const MAX_CHAR = 32;                            { Maximum number of redefinable characters }

const UseChars : array[1..MAX_CHAR] of byte =    { Redefinable chars. }
( 128, 130, 131, 133, 134, 135, 136, 137, 138, 139,
  140, 141, 143, 144, 145, 146, 147, 149, 150, 151,
  152, 155, 156, 157, 158, 159, 160, 161, 162, 163,
  164, 165 );

var Regs      : Registers;    { Processor registers for interrupt call }
    videoc    : CRDTYPE;      { Type of video card installed }
    chardef    : array[0..15] of byte; { Bit pattern of one character }
    charheight, { Number of scan lines per character }
    i, j, k, l, { Loop variables }
    bmask,      { Bit mask for generating a scan line }
    swidth,     { String width }
    index,      { Index for executing the UseChars array }
    dx,         { Logo block width (text columns) }
    dy,         { Logo block depth (text rows) }
    leftb,      { Left border in pixels }
    rightb,     { Right border in pixels }
    topb,       { Top border in pixels }
    bttmb      : byte;        { Bottom border in pixels }
    bptr       : barptr;      { For addressing logo buffer }

{-- IsSet function: Checks for set logo pixels -----}

function IsSet( lgrow, column : byte ) : boolean;

begin
    if ( lgrow < topb ) or ( lgrow > bttmb ) or { Pixel outside }
      ( column < leftb ) or ( column > rightb ) then { border? }
        IsSet := false { Yes --> Don't set it }
    else { No --> Pass to logo buffer }
        IsSet := bptr^[ (lgrow-topb)*(swidth+1) + 1 + (column-leftb) ] <> 32;
    end;

{-- Main procedure -----}

begin
    videoc := IsEgaVga; { Check for video card }
    case videoc of
        NEITHERNOR :
            begin
                writeln( 'Warning: No EGA or VGA card found' );
                exit;
            end;
        EGA :
            charheight := 14; { EGA: 14 scan lines per character }
        VGA :
            begin
                { VGA }
                SetCharWidth( 8 ); { Set char. width to 8 pixels }
                charheight := 16; { 16 scan lines per character }
            end;
    end;

    bptr := @buf; { Set pointer to logo buffer }
    swidth := bptr^[0]; { Get string length and logo width }
    dx := ( swidth + 7 ) div 8; { Compute number of characters needed }
    dy := ( depth + charheight - 1 ) div charheight;
    if ( dx*dy > MAX_CHAR ) then
        writeln( 'Error: Logo in BuildLogo too large' )

```

```

else
begin
    topb    := ( dy*charheight-depth ) div 2;           { Compute border }
    bttmb   := depth + topb - 1;
    leftb   := ( dx*8-swidth ) div 2;
    rightb  := swidth + leftb - 1;

    TextColor( opcol and 15 );                           { Set display color }
    TextBackGround( opcol shr 4 );
    index := 1;                                           { Get first character from UseChar array }
    for i := 0 to dy-1 do                                { Execute text rows }
    begin
        GotoXY( column, lgrow + i );
        for j := 1 to dx do                              { Execute text characters }
        begin
            write( chr( UseChars[ index ] ) ); { Display character }

            {-- Compute new character pattern for the char -----}

            for k := 0 to charheight-1 do                { Execute scan lines }
            begin
                bmask := 0;                               { Bit mask orig. 0 }
                for l := 0 to 7 do { Each character is 8 pixels wide }
                begin
                    bmask := bmask shl 1;                 { Move mask left }
                    if IsSet( i*charheight+k, (j-1)*8+1 ) then
                        bmask := bmask or 1;              { Set pixel in logo }
                    end;
                    chardef[ k ] := bmask; { Bit pattern in char. buffer }
                end;
                Regs.AX := $1100;                          { Call BIOS video interrupt, }
                Regs.BH := charheight;                     { function 10H, sub-function }
                Regs.BL := 0;                               { 00H to specify new }
                Regs.CX := 1;                               { character pattern }
                Regs.DX := UseChars[ index ];
                Regs.ES := seg( chardef );
                Regs.BP := ofs( chardef );
                intr( $10, Regs );

                inc( index );                               { Get next character from UseChar }
            end;
        end;
    end;
end;

{ *****
* ResetLogo : Reloads original character definitions.
* -----
* Input : None
* ***** }

procedure ResetLogo;

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

begin
    case IsEgaVga of
        EGA : begin
            Regs.AX := $1101;                               { Load new 8x14 font }
            Regs.BL := 0;
            intr( $10, Regs );
            end;

        VGA : begin
            SetCharWidth( 9 );                               { Set char. width to 9 pixels }
            Regs.AX := $1104;                               { Load new 8x16 font }
            Regs.BL := 0;
            intr( $10, Regs );
            end;
    end;
end;

{ *****
* Test : Demonstrates the BuildLogo procedure.
* -----
* Input : None
* ***** }

procedure Test;

const MyLogo : array[1..32] of string[38] =
    (
        '          ***',
        '          ****',
        '          ****',
        '          **'
    );

```

```
begin
    Test;
end.
```