

```

{*****
*
*                               V 1 6 C O L P . P A S
*
**-----**
*   Task                       : Demonstrates programming in EGA and VGA graphic
*                               modes using 16 colors. This program requires
*                               the V16COLPA.ASM assembly language module.
*
**-----**
*   Author                     : MICHAEL TISCHER
*   Developed on                : 12/20/90
*   Last update                 : 01/14/91
*****}

```

```

program V16COLP;

```

```

uses dos, crt;

```

```

{-- Type declarations -----}

```

```

type BPTR = ^byte;

```

```

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

```

```

{$L v16colpa}                                { Link assembler module }

```

```

procedure init640480; external;
procedure init640350; external;
procedure init320200; external;
procedure init640200; external;
procedure setpix( x, y : integer; pcolor : byte ); external;
function  getpix( x, y: integer ) : byte; external;
procedure setpage( page : integer ); external;
procedure showpage( page : integer ); external;

```

```

{-- Constants -----}

const A320200 = 1;           { Possible resolutions and modes }
    A640200 = 2;
    A640350 = 3;
    A640480 = 4;

    VMODE = A640350;        { Specify constants for desired mode }
                             { here }

{-- Type declarations -----}

type VCARD = ( EGA, VGA, NEITHERNOR ); { Type of installed video card }

{-- Global variables -----}

var MaxX,                    { Maximum X- and Y-coordinates }
    MaxY : integer;
    Pages : byte;           { Number of screen pages }

{*****
* IsEgaVga : Determines whether an EGA or a VGA card is installed. *
*****}
*-----*
* Input : None *
* Output : TRUE if EGA or VGA card, otherwise FALSE *
*****}

function IsEgaVga : VCARD;

var Regs : Registers;      { Processor 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 a 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;

{ *****
*   Line: Draws a line based on the Bresenham algorithm.           *
*   -----*
*   Input   : X1, Y1 = Starting coordinates (0 - ...)             *
*             X2, Y2 = Ending coordinates                         *
*             LPCOL = Color of the 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-axis }
      if ( y1 > y2 ) then                  { y1 > y2? }
        begin
          SwapInt( x1, x2 );              { Yes -->   swap X1 with Y1 }
          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
    { Check X-axes }
begin
    if ( x1 > x2 ) then
        { x1 > x2? }
        begin
            SwapInt( x1, x2 );
            { Yes --> swap X1 with X2 }
            SwapInt( y1, y2 );
            { and Y1 with Y2 }
        end;

        if ( y2 > y1 ) then yincr := 1
            { Set Y-axis increment }
            else yincr := -1;

        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 to x2 do
            { Execute line on X-axes }
            begin
                if ( d >= 0 ) then
                    begin

```

```

        inc( y, yincr );
        inc( d, aincr );
    end
else
    inc( d, bincr );
    setpix( x, y, lpcol );
end;
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;

{*****}

```



```

*   ColorBox:  Draws a rectangle and fills it with a line pattern.      *
**-----**
*   Input    :  X1, Y1 = Upper-left coordinates of window              *
*               X2, Y2 = Lower-right coordinates of window            *
*               COLMAX = Greatest color value                          *
*   Info     :  Line colors are selected in a cycle of 0-COLMAX        *
*****}

```

```

procedure ColorBox( x1, y1, x2, y2 : integer; colmax : byte );

var x, y,
    sx, sy : integer;
                                { Loop counter }
                                { Exit point for last color loop }

begin
    Line( x1, y1, x1, y2, 15 );
                                { Draw border }
    Line( x1, y2, x2, y2, 15 );
    Line( x2, y2, x2, y1, 15 );
    Line( x2, y1, x1, y1, 15 );

    for y := y2-1 downto y1+1 do
        Line( x1+1, y2-1, x2-1, y, y mod colmax );
                                { Bottom left to right border }

    for y := y2-1 downto y1+1 do
        Line( x2-1, y2-1, x1+1, y, y mod colmax );
                                { Bottom right to left border }

    {-- From center of box to top border -----}

    sx := x1+ (x2-x1) div 2;
    sy := y1+ (y2-y1) div 2;
    for x := x1+1 to x2-1 do
        Line( sx, sy, x, y1+1, x mod colmax );
    end;

```

```

{*****
* DrawAxis: Draws axes from left and top borders on the screen. *
**-----**
* Input : STEPX = Increment for X-axis *
*        STEPY = Increment for Y-axis *
*        FG = Foreground color *
*        BK = Background color (255 = transparent) *
*****}

```

```

procedure DrawAxis( stepx, stepy : integer; fg, bk : byte );

```

```

var x, y      : integer;           { Loop coordinates }
    ordinate : string[3];

```

```

begin

```

```

    Line( 0, 0, MAXX, 0, fg );      { Draw X-axis }
    Line( 0, 0, 0, MAXY, fg );      { Draw Y-axis }

```

```

    x := stepx;                      { Scale X-axis }

```

```

    while ( x < MAXX ) do

```

```

        begin

```

```

            Line( x, 0, x, 5, fg );

```

```

            str( x, ordinate );

```

```

            if ( x < 100 ) then

```

```

                GrfxPrint( x - 8 , 8, fg, bk, ordinate )

```

```

            else

```

```

                GrfxPrint( x - 12, 8, fg, bk, ordinate );

```

```

            inc( x, stepx );

```

```

        end;

```

```

    y := stepy;                      { Scale Y-axis }

```

```

while ( y < MAXY ) do
  begin
    Line( 0, y, 5, y, fg );
    str( y:3, ordinate );
    GrfxPrint( 8, y-4, fg, bk, ordinate );
    inc( y, stepy );
  end;
end;

{ *****
* Demo: Demonstrates the functions and procedures in this module. *
**-----**
* Input      : None *
*****}

procedure Demo;

const PAUSE = 100;                { Pause counter in milliseconds }

var pgcount : byte;               { Page counter }

begin
  for pgcount := 1 to Pages do
    begin
      setpage( pgcount-1 );        { Process page }
      showpage( pgcount-1 );      { Process page }
      ColorBox( 50+pgcount*2, 40, MaxX-50+pgcount*2, MaxY-40, 16 );
      DrawAxis( 30, 20, 15, 255 ); { Draw axes }
      GrfxPrint( 46, MAXY-10, 15, 255,
        'V16COLP - (c) by Michael Tischer' );
    end;
  end;

```

```

{-- Display graphic pages in sequences -----}

for pgcount := 0 to 50 do                                { 50 executions }
begin
    showpage( pgcount mod Pages );                        { Display page }
    delay( PAUSE );                                       { Brief pause }
end;
end;

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

begin
writeln( 'V16COLP.PAS - (c) 1992 by Michael Tischer'#13#10 );
if ( VMODE = A640480 ) then                                { VGA mode selected? }
begin                                                       { Yes }
    if ( IsEgaVga <> VGA ) then                                { VGA card installed? }
    begin                                                    { No }
        writeln( 'This program requires a VGA card' );
        exit;                                                { End program }
    end
else                                                         { Yes --> initialize mode and set parameters }
begin                                                         { 640x480 pixels }
    MaxX := 639;
    MaxY := 479;
    Pages := 1;
    init640480;
end;
end
else                                                         { Must be one of the EGA modes }
begin

```

