

```

***** V I D E O P *****
*
* Task      : Makes functions available based on the BIOS
*            video interrupt not provided by Pascal.
*
*-----
* Author     : Michael Tischer
* Developed on : 07/10/87
* Last update  : 02/18/92
*****

```

```

program VIDEOP;

```

```

Uses Crt, Dos;                                { Add DOS and CRT units }

```

```

const NORMAL      = $07;    { Definition of character attribute }
      BOLD         = $0f;    { in relation to a Monochrome      }
      INVERS       = $70;    { Display Adapter                  }
      UNDERLINE    = $01;
      BLINK        = $80;

```

```

type TextTyp = string[80];

```

```

var i,                { Loop variable for the main program }
    j,
    k,
    l      : integer;
    IString : string[2];    { Accepts number of arrows }

```

```

*****
* GETVIDEOMODE: Reads current video mode and parameters.
* Input       : None
* Output      : Variables receive values after procedure call
*****

```

```

procedure GetVideoMode(var VideoMode, { Number of current video mode }
                      Number,         { Number of columns per line }
                      Page : integer); { Current screen page }

```

```

var Regs : Registers;    { Register variables for interrupt call }

```

```

begin
  Regs.ah := $0F;    { Function number }
  intr($10, Regs);   { Call BIOS video interrupt }
  VideoMode := Regs.al; { Number of video mode }
  Number := Regs.ah;   { Number of characters per line }
  Page := Regs.bh;     { Number of the current screen page }
end;

```

```

*****
* SETCURSORATYPE: Defines the appearance of the blinking cursor.
* Input          : See below
* Output         : None
* Info           : Parameters can be from 0 to 13 for a Monochrome
*                  Display Adapter, and from 0 to 7 for a color card.
*****

```

```

procedure SetCursorType(Beginline, { Beginning line of the cursor }
                       Endl : integer); { End line of the cursor }

```

```

var Regs : Registers;    { Register variable for the interrupt call }

```

```

begin
  Regs.ah := 1;    { Function number }
  Regs.ch := Beginline; { Beginning and }
  Regs.cl := Endl;   { End line }
  intr($10, Regs);   { Call BIOS video interrupt }
end;

```

```

*****
* SETCURSORPOS: Defines cursor position during screen page display.
* Input        : See below
* Output       : None
* Info         : The cursor position changes only when this
*                  procedure is called, if the current screen page is
*                  indicated.
*****

```

```

procedure SetCursorPos(Page, { Screen page containing cursor }
                      Column, { New cursor column }
                      CRow : integer); { New cursor row }

```

```

var Regs : Registers;    { Register variable for the interrupt }

```

```

begin

```

```

Regs.ah := 2; { Function number }
Regs.bh := Page; { Screen page }
Regs.dh := CRow; { Display coordinates }
Regs.dl := Column;
intr($10, Regs); { Call BIOS video interrupt }
end;

{*****}
{ * GETCURSORPOS: Gets the cursor's position, starting and ending * }
{ * lines. * }
{ * Input : See below * }
{ * Output : Variables contain values after procedure call * }
{ * Info : Start and end cursor lines are independent of the * }
{ * screen page. * }
{*****}

procedure GetCursorPos(Page : integer; { The screen page }
var Column, { Cursor column }
CRow, { Cursor row }
Beginline, { Start line of the cursor }
Endl : integer); { End line of the cursor }

var Regs : Registers; { Register variables for the interrupt }

begin
Regs.ah := 3; { Function number }
Regs.bh := Page; { Screen page }
intr($10, Regs); { Call BIOS video interrupt }
Column := Regs.dl; { Read function results }
CRow := Regs.dh; { from these registers }
Beginline := Regs.ch; { and store in proper }
Endl := Regs.cl; { variables }
end;

{*****}
{ * SETSCREENPAGE : Sets the screen page for output on the monitor. * }
{ * Input : See below * }
{ * Output : None * }
{*****}

procedure SetScreenPage(Page : integer); { The new screen page }

var Regs : Registers; { Register variables for interrupt call }

begin
Regs.ah := 5; { Function number and screen page }
Regs.al := Page; { Screen page }
intr($10, Regs); { Call BIOS video interrupt }
end;

{*****}
{ * SCROLLUP: Scrolls a display area by one or more * }
{ * lines up or erases it. * }
{ * Input : See below * }
{ * Output : None * }
{ * Info : If number 0 is passed, the display area * }
{ * is filled with spaces * }
{*****}

procedure ScrollUp(Number, { Number of lines to be scrolled }
COLOR, { Attribute for the blank lines created }
ColumnUL, { Column in the upper-left corner }
CRowUL, { Row in the upper-left corner }
ColumnLR, { Column in the lower-right corner }
CRowLR : integer); { Row in the lower-right corner }

var Regs : Registers; { Register variables for interrupt call }

begin
Regs.ah := 6; { Function number and number }
Regs.al := Number;
Regs.bh := COLOR; { Color of empty line(s) }
Regs.ch := CRowUL; { Upper-left }
Regs.cl := ColumnUL; { coordinates }
Regs.dh := CRowLR; { Lower-right }
Regs.dl := ColumnLR; { coordinates }
Intr($10, Regs); { Call BIOS video interrupt }
end;

{*****}
{ * SCROLLDOWN: Scrolls a display area by one or more * }
{ * lines down or erases it. * }
{ * Input : See below * }
{ * Output : None * }
{ * Info : If number 0 is passed, the display area * }
{ * is filled with spaces. * }
{*****}

```

```

{*****}
procedure ScrollDown(Number,           { Number of lines to be scrolled }
                    COLOR, { Attribute for the blank line(s) created }
                    ColumnUL,       { Column in the upper-left corner }
                    CRowUL,         { Row in the upper-left corner }
                    ColumnLR,       { Column in the lower-right corner }
                    CRowLR : integer); { Row in lower-right corner }

var Regs : Registers;           { Register variables for interrupt call }

begin
  Regs.ah := 7;                  { Function number and number }
  Regs.al := Number;
  Regs.bh := COLOR;              { Color of blank line(s) }
  Regs.ch := CRowUL;             { Upper-left }
  Regs.cl := ColumnUL;           { coordinates }
  Regs.dh := CRowLR;             { Lower-right }
  Regs.dl := ColumnLR;           { coordinates }
  Intr($10, Regs);              { Call BIOS video interrupt }
end;

{*****}
{ * GETCHAR: Reads a character including attribute from an indicated * }
{ * position in a screen page. * }
{ * Input : See below * }
{ * Output : See below * }
{*****}

procedure GetChar(Page,                { Screen page accessed }
                  Column,              { Screen column }
                  SRow : integer;      { Screen row }
                  var Character : char; { Character }
                  var COLOR : integer); { Its attribute }

var Regs : Registers;           { Register variables for interrupt call }
  CurColumn,                    { Current column }
  CurCRow,                      { Current row }
  CurPage,                      { Current screen page }
  Dummy : integer;              { Stores unnecessary variables }

begin
  GetVideoMode(Dummy, Dummy, CurPage); { Get current screen page }
  GetCursorPos(CurPage, CurColumn, CurCRow, { Get cursor position in }
              Dummy, Dummy); { the current screen page }
  SetCursorPos(Page, Column, SRow);      { Set cursor position }

  Regs.ah := 8;                  { Get function number for char. and attribute }
  Regs.bh := Page;              { Screen page }
  Intr($10, Regs);              { Invoke DOS registers }
  Character := chr(Regs.al);     { ASCII character code }
  COLOR := Regs.ah;             { Character attribute }
  SetCursorPos(CurPage, CurColumn, CurCRow); { Set old cursor position }
end;

{*****}
{ * WRITECHAR: Writes a character with indicated color to the * }
{ * current cursor position in the screen page. * }
{ * Input : See below * }
{ * Output : None * }
{ * Info : During execution, control characters (CRLF) are * }
{ * handled as control characters. * }
{*****}

procedure WriteChar(Page : integer; { Screen page for writing }
                   Character : char; { ASCII character code }
                   COLOR : integer); { Its attribute }

var Regs : Registers;           { Register variables for interrupt call }

begin
  Regs.ah := 9;
  Regs.al := ord(Character);     { Function number and character code }
  Regs.bh := Page;              { Screen page }
  Regs.bl := COLOR;             { Display color }
  Regs.cx := 1;                 { Display character only once }
  Intr($10, Regs);              { Call BIOS video interrupt }
end;

{*****}
{ * WRITETEXT: Writes a string starting at an indicated position in * }
{ * a screen page. * }
{ * Input : See below * }
{ * Output : None * }
{ * Info : During execution, control characters (CRLF) are * }
{ * handled as control characters. If display goes past * }

```

```

the last screen line, the display scrolls up.
{*****}
procedure WriteText(Page,           { Screen page for output }
                   Column,         { Column, from which output starts }
                   SRow,           { Line, from which output starts }
                   COLOR : integer; { Color for all characters }
                   Text : TextTyp); { Text for output }

var Regs : Registers;      { Register variables for interrupt call }
    Counter : integer;     { Loop counter }

begin
  SetCursorPos(Page, Column, SRow); { Set cursor }

  for Counter := 1 to length(Text) do { Process characters }
  begin                               { in sequence }
    WriteChar(Page, ' ', COLOR);      { Color at the current position }
    Regs.ah := 14;
    Regs.al := ord(Text[Counter]);    { Function number and character }
    Regs.bh := Page;                  { Screen page }
    Intr($10, Regs);                  { Call BIOS video interrupt }
  end;
end;

{*****}
**                               MAIN PROGRAM                               **
{*****}

begin
  clrscr;                          { Clear screen }
  for i := 1 to 24 do               { Do rows 1 to 24 }
  for j := 0 to 79 do               { Do all columns }
  begin
    SetCursorPos(0, j, i);          { Position cursor }
    WriteChar(0, chr(i*80+j and 255), NORMAL); { Write a character }
  end;
  ScrollDown(0, NORMAL, 5, 8, 19, 22); { Clear window 1 }
  WriteText(0, 5, 8, INVERS, ' Window 1 ');
  ScrollDown(0, NORMAL, 60, 2, 74, 16); { Clear window 2 }
  WriteText(0, 60, 2, INVERS, ' Window 2 ');
  WriteText(0, 24, 12, INVERS or BLINK, ' >>> PC INTERN <<< ');
  WriteText(0, 0, 0, INVERS, 'ing in sequence --> <-- arrows remain'+
  'ing in sequence ');
  for i := 49 downto 0 do           { Draw a total of 50 arrows }
  begin
    str(i:2, IString);              { Convert i to ASCII string }
    WriteText(0, 20, 0, INVERS, IString);
    j := 1;                         { Every arrow consists of 16 lines }
    while j <= 15 do
    begin
      k := 0;
      while k < j do                { Create a line for an arrow }
      begin
        SetCursorPos(0, 12-(j shr 1)+k, 9); { Arrow window 1 }
        WriteChar(0, '*', BOLD);
        SetCursorPos(0, 67-(j shr 1)+k, 16); { Arrow window 2 }
        WriteChar(0, '*', BOLD);
        k := succ(k);
      end;
      ScrollDown(1, NORMAL, 5, 9, 19, 22); { Scroll window 1 }
      ScrollUp(1, NORMAL, 60, 3, 74, 16); { Scroll window 2 }
      for l := 0 to 8000 do          { Delay loop }
      ;
      j := j+2;
    end;
  end;
  clrscr;                          { Clear screen }
end.

```