


```

void GetVideoMode(VideoMode, Number, Page)
int *VideoMode;                /* Number of current video mode */
int *Number;                    /* Number of columns per line */
int *Page;                      /* Current screen page */

{
    union REGS Register;        /* Register variables for interrupt call */

    Register.h.ah = 15;         /* Function number */
    int86(0x10, &Register, &Register); /* Call interrupt 10H */
    *VideoMode = Register.h.al; /* Number of video mode */
    *Number = Register.h.ah;     /* Number of characters per line */
    *Page = Register.h.bh;      /* Number of current screen page */
}

/*****
/* SETCURSOR: 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. */
*****/

void SetCursorType(Beginline, Endl)
int Beginline;                /* Beginning line of the cursor */
int Endl;                      /* End line of the cursor */

{
    union REGS Register;        /* Register variables for interrupt call */

    Register.h.ah = 1;         /* Function number */

```

```

Register.h.ch = Beginline;          /* Beginning line of cursor */
Register.h.cl = Endl;               /* End line of cursor */
int86(0x10, &Register, &Register); /* Call interrupt 10H */
}

/*****
/* 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. */
*****/

void SetCursorPos(Page, Column, CRow)
int Page;          /* Screen page containing cursor */
int Column;        /* New cursor column */
int CRow;          /* New cursor row */

{
    union REGS Register; /* Register variables for interrupt call */

    Register.h.ah = 2;    /* Function number */
    Register.h.bh = Page; /* Screen page */
    Register.h.dh = CRow; /* Screen row */
    Register.h.dl = Column; /* Screen column */
    int86(0x10, &Register, &Register); /* Call interrupt 10H */
}

/*****
/* GETCURSORPOS: Gets the cursor's position, starting and ending */
/*             lines. */
*****/

```



```

{
    union REGS Register;          /* Register variables for interrupt call */

    Register.h.ah = 5;            /* Function number */
    Register.h.al = Page;         /* Screen page */
    int86(0x10, &Register, &Register); /* Call interrupt 10H */
}

```

```

/*****
/* SCROLLUP: Scrolls a screen area up one or several          */
/*          lines or erases it.                                */
/* Input    : See below                                       */
/* Output   : None                                           */
/* Info     : If number 0 is passed, the screen area         */
/*           is filled with spaces.                           */
*****/

```

```

void ScrollUp(Number, Color, ColumnUL, CRowUL, ColumnLR, CRowLR)
int Number;          /* Number of lines to be scrolled */
int Color;           /* Color or attribute for the blank lines */
int ColumnUL;        /* Column in upper-left corner */
int CRowUL;          /* Row in upper-left corner */
int ColumnLR;        /* Column in lower-right corner */
int CRowLR;          /* Row in lower-right corner */

```

```

{
    union REGS Register;          /* Register variables for interrupt call */

    Register.h.ah = 6;            /* Function number */
    Register.h.al = Number;       /* Number of lines */
    Register.h.bh = Color;        /* Color of blank line(s) */
    Register.h.ch = CRowUL;       /* Set coordinates of the */
}

```

```

Register.h.cl = ColumnUL;                /* window to be scrolled */
Register.h.dh = CRowLR;                  /* or erased */
Register.h.dl = ColumnLR;
int86(0x10, &Register, &Register);      /* Call interrupt 10H */
}

```

```

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

```

```

void ScrollDown(Number, Color, ColumnUL, CRowUL, ColumnLR, CRowLR)
int Number;                /* Number of lines to be scrolled */
int Color;                 /* Color or attribute for the blank lines */
int ColumnUL;              /* Column in upper-left corner */
int CRowUL;                /* Row in upper-left corner */
int ColumnLR;              /* Column in lower-right corner */
int CRowLR;                /* Row in lower-right corner */

```

```

{
    union REGS Register;    /* Register variables for interrupt call */

    Register.h.ah = 7;      /* Function number */
    Register.h.al = Number; /* Number of lines */
    Register.h.bh = Color;  /* Color of blank line(s) */
    Register.h.ch = CRowUL; /* Set coordinates for the */
    Register.h.cl = ColumnUL; /* window to be scrolled */
    Register.h.dh = CRowLR;  /* or erased */
}

```

```

Register.h.dl = ColumnLR;
int86(0x10, &Register, &Register);          /* Call interrupt 10H */
}

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

void GetChar(Page, Column, SRow, Character, Color)
int Page;          /* Screen page accessed */
int Column;        /* Screen column */
int SRow;          /* Screen row */
char *Character;   /* Character at this position */
int *Color;        /* Its attribute byte (color) */

{
union REGS Register; /* Register variables for interrupt call */
int Dummy;           /* Stores unnecessary variables */
int CurPage;         /* Current screen page */
int CurCRow;         /* Current row */
int CurColumn;       /* Current column */

GetVideoMode(&Dummy, &Dummy, &CurPage); /* Get current screen page */
GetCursorPos(&CurPage, &CurColumn, &CurCRow, /* Get current */
             &Dummy, &Dummy); /* cursor position */
SetCursorPos(Page, Column, SRow); /* Set cursor */
Register.h.ah = 8; /* Function number */
Register.h.bh = Page; /* Screen page */
int86(0x10, &Register, &Register); /* Call interrupt 10H */
}

```

```

*Character = Register.h.al;          /* Read results from the */
*Color = Register.h.ah;              /* registers and assign */
SetCursorPos(CurPage, CurColumn, CurCRow); /* Set old cursor pos. */
}

/*****
/* WRITECHAR: Writes a character with indicated color to the */
/*          current cursor position in the screen page.      */
/* Input    : See below                                     */
/* Output   : None                                           */
*****/

void WriteChar(Page, Character, Color)
int Page;          /* Screen page for writing */
char Character;    /* ASCII character code */
int Color;         /* Its attribute or color */

{
    union REGS Register; /* Register variables for interrupt call */

    Register.h.ah = 9;    /* Function number */
    Register.h.al = Character; /* Character code */
    Register.h.bh = Page;  /* Screen page */
    Register.h.bl = Color; /* Character color */
    Register.x.cx = 1;     /* Display character only once */
    int86(0x10, &Register, &Register); /* Call interrupt 10H */
}

/*****
/* WRITETEXT: Writes a string starting at a specific screen position.*/
/* Input    : See below                                     */
/* Output   : None                                           */
*****/

```



```

/* Info      : Text is a pointer to a character vector which contains */
/*            the text to be output and is terminated                  */
/*            with a '\0' character.                                   */
/******/

void WriteText(Page, Column, SRow, Color, Text)
int Page;                                /* Screen page for output */
int Column;                             /* Column for output */
int SRow;                               /* Row for output */
int Color;                             /* Color for all characters */
char *Text;                             /* Text for output */

{
    union REGS Register;                /* Register variables for interrupt call */

    SetCursorPos(Page, Column, SRow);   /* Set cursor */
    while (*Text)                       /* Process text up to '\0' character */
    {
        WriteChar(Page, ' ', Color);    /* Character color */
        Register.h.ah = 14;              /* Function number */
        Register.h.bh = Page;            /* Screen page */
        Register.h.al = *Text++;         /* Character */
        int86(0x10, &Register, &Register); /* Call interrupt */
    }
}

/*****/
/* CLEARSCREEN: Clears the 80x25 text screen. */
/* Input      : None */
/* Output     : None */
/*****/

```

```

void ClearScreen()

{
    int CurPage;                                /* Current screen page */
    int Dummy;                                  /* Dummy variable */

    ScrollUp(0, NORMAL, 0, 0, 79, 24);          /* Clear screen */
    GetVideoMode(&Dummy, &Dummy, &CurPage);    /* Get current screen page */
    SetCursorPos(CurPage, 0, 0);                /* Set cursor */
}

/*****
**                                     MAIN PROGRAM                                     **
*****/

void main()

{
    int i, j, k, l;                             /* Loop variables */
    char Arrows[3];                             /* accepts number of arrows as ASCII string */

    ClearScreen();                              /* Clear screen */
    for (i = 1; i < 25; i++)                    /* Process all rows */
        for (j = 0; j < 80; j++)                /* Process all columns */
        {
            SetCursorPos(0, j, i);              /* Position cursor */
            WriteChar(0, i*80+j&255, NORMAL);    /* Write characters */
        }
    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 | BLINK, " >>>      PC      INTERN      <<< ");
WriteText(0, 0, 0, INVERS, "      -->      <-- arrows remain");
WriteText(0, 40, 0, INVERS, "ing in sequence      ");
for (i = 49; i >= 0 ; i--)          /* Draw 50 arrows */
{
    sprintf(Arrows, "%2d", i);      /* Convert number of arrows to ASCII */
    WriteText(0, 20, 0, INVERS, Arrows);          /* and output */
    for (j = 1; j < 16; j+= 2)      /* Every arrow consists of 16 lines */
    {
        for (k = 0; k < j; k++)          /* Create a line for an arrow */
        {
            SetCursorPos(0, 12-(j>>1)+k, 9);          /* Arrow window 1 */
            WriteChar(0, '*', BOLD);
            SetCursorPos(0, 67-(j>>1)+k, 16);          /* Arrow window 2 */
            WriteChar(0, '*', BOLD);
        }
        ScrollDown(1, NORMAL, 5, 9, 19, 22);          /* Scroll window 1 down */
        ScrollUp(1, NORMAL, 60, 3, 74, 16);          /* Scroll window 2 up */
        for (l = 0; l < 4000 ; l++)          /* Wait loop */
            ;
    }
}
ClearScreen();          /* Clear screen */
}

```