

```

/*****
/*
/*----- M O U S E C . C -----*/
/*
/* Task : Demonstrates mouse access from the C language */
/*-----*/
/*
/* Author : MICHAEL TISCHER */
/* Developed on : 04/20/1989 */
/* Last update : 04/07/1995 */
/*-----*/
/*
/* Microsoft C */
/* Creation : CL /AS MOUSEC.C MOUSECA.OBJ */
/* Call : MOUSEC */
/*-----*/
/*
/* Turbo C (integrated system) */
/* Creation : Create a project file containing the following:*/
/*
/* MOUSEC */
/*
/* MOUSECA.OBJ */
/*
/* Make sure that memory model is set to small. */
/*
/* If you didn't assemble the MOUSECA.ASM file */
/*
/* using the /MX option in MASM, make sure that */
/*
/* Case-Sensitive Link on Linker options is OFF. */
/*
/* Disable stack checking before compilation. */
/*
/* >>NOTE: One warning will occur (about the */
/*
/* ButState in the MouEventHandler function). */
/*
/* The program will run. Do N O T remove */
/*
/* the ButState declaration - the AssmHand routine*/
/*
/* needs it<< */
/*
/* Call : MOUSEC */
/*-----*/
/*****

/*== Add include files =====*/

#include <dos.h>
#include <stdlib.h>

extern void far AssmHand( void ); /* External declaration */
/* of assembler handler */

/*== Typedefs =====*/

typedef unsigned char BYTE; /* Create a byte */
typedef unsigned long PTRVIEW; /* Mouse pointer mask */
typedef struct { /* Describe a mouse range */
    BYTE x1, /* Upper left coordinates of the */
        y1, /* specified range */
    x2, /* Lower right corner of the */
        y2; /* specified range */
    PTRVIEW ptr_mask; /* Mouse pointer mask */
} RANGE;
typedef void (far * MOUHAPTR)( void ); /* Pointer to event handler */

/*== Constants =====*/

#define TRUE ( 1 == 1 )
#define FALSE ( 1 == 0 )

/*-- Event codes -----*/

#define EV_MOU_MOVE 1 /* Move mouse */
#define EV_LEFT_PRESS 2 /* Left mouse button pressed */
#define EV_LEFT_REL 4 /* Left mouse button released */
#define EV_RIGHT_PRESS 8 /* Right mouse button pressed */
#define EV_RIGHT_REL 16 /* Right mouse button released */
#define EV_MOU_ALL 31 /* all mouse events */

#define NO_RANGE 255 /* mouse pointer not in range xy */

/*-- Macros -----*/

#define MouGetCol() (ev_col) /* Return mouse position & */
#define MouGetRow() (ev_row) /* range the moment the */
#define MouGetRange() (ev_rng) /* event occurs */
#define MouAvail() (mavail) /* Available mouse = TRUE */
#define MouGetCurCol() (moucol) /* Returns current mouse */
#define MouGetCurRow() (mourow) /* position and current */
#define MouGetCurRng() (mourng) /* mouse range */
#define MouIsLeftPress() (mouevent & EV_LEFT_PRESS )

```

```

#define MouIsLeftRel()      ( mouevent & EV_LEFT_REL )
#define MouIsRightPress()   ( mouevent & EV_RIGHT_PRESS )
#define MouIsRightRel()     ( mouevent & EV_RIGHT_REL )
#define MouSetMoveAreaAll() MouSetMoveArea( 0, 0, tcol-1, tline-1 );

#define ELVEC(x) ( sizeof(x) / sizeof(x[0]) ) /* No. of elements in X */

/*-- Bitmask creation macros defining mouse pointer's appearance. ---*/
/*-- Syntax for calling MouPtrMask (sample): ---*/
/*-- MouPtrMask( PTRDIFCHAR( 'x' ), PTRINVCOL ) ---*/
/*-- When the pointer is represented as a lowercase x, the inverse ---*/
/*-- character color takes effect. ---*/

#define MouPtrMask( z, f )\
    ( (( (PTRVIEW) f) >> 8 << 24) + ((( PTRVIEW) z) >> 8 << 16) +\
      (((f) & 255) << 8) + ((z) & 255) )

#define PTRSAMECHAR ( 0x00ff ) /* Same character */
#define PTRDIFCHAR(z) ( (z) << 8 ) /* Other characters */
#define PTRSAMECOL ( 0x00ff ) /* Same color */
#define PTRINVCOL ( 0x7777 ) /* Inverse color */
#define PTRSAMECOLB ( 0x807f ) /* Same color (blinking) */
#define PTRINVCOLB ( 0xF777 ) /* Inverse color (blinking) */
#define PTRDIFCOL(f) ( (f) << 8 ) /* Other color */
#define PTRDIFCOLB(f) (((f)|0x80) << 8) /* Other color (blinking) */

#define EAND 0 /* Event comparisons for MouEventWait() */
#define EVOR 1

#define MOUINT(rin, rout) int86(0x33, &rin, &rout)
#define MOUINTX(rin, rout, sr) int86x(0x33, &rin, &rout, &sr)

/*-- Macros for converting mouse coordinates between virtual mouse */
/*-- screen and text screen ----*/

#define XTOCOL(x) ( (x) >> 3 ) /* X v 8 */
#define YTOROW(y) ( (y) >> 3 ) /* Row v 8 */
#define COLTOX(c) ( (c) << 3 ) /* C x 8 */
#define ROWTOY(r) ( (r) << 3 ) /* Row x 8 */

/*== global variables =====*/
BYTE tline, /* No. of text lines */
tcol, /* No. of text columns */
mavail = FALSE; /* TRUE when mouse is available */

/*-- Mask for standard mouse pointer -----*/
PTRVIEW stdptr = MouPtrMask( PTRSAMECHAR, PTRINVCOL );

BYTE * bbuf, /* Ptr to range recognition buffer */
num_range = 0; /* No range defined until now */

RANGE * cur_range; /* Pointer to current range vector */
int blen; /* Length of BBUF in bytes */

/*-- Variables which load every time the mouse handler is called ----*/
BYTE mourng = NO_RANGE, /* Current mouse range */
moucol, /* Mouse column (text screen) */
mourow; /* Mouse row (text screen) */
int mouevent = EV_LEFT_REL + EV_RIGHT_REL; /* Event mask */

/*-- Variables which load every time an event anticipated by the ---*/
/*-- mouse handler occurs ----*/

BYTE ev_rng, /* Range in which the mouse can be found */
ev_col, /* Mouse column */
ev_row; /* Mouse row */

/*****
* Function : M o u D e f i n e P t r
*-----*
* Task : Defines the cursor mask and screen mask which
* determines the mouse pointer's appearance
* Input parameters : MASK = Both bitmasks, made into a 32-bit value
*****/

```

```

*                                     of type UNSIGNED LONG                                     *
* Return value      : None                                                    *
* Info              : Most significant 16 bits of MASK = screen mask          *
*                   : least significant 16 bits of mask = cursor mask          *
*****
#pragma check_stack(off)                                     /* No stack checking here */

void MouDefinePtr( PTRVIEW mask )
{
    static PTRVIEW oldercursor = (PTRVIEW) 0;      /* Last value for MASK */
    union REGS regs;                               /* Processor regs for interrupt call */

    if ( oldercursor != mask )                     /* Changes since last call? */
    {                                               /* YES */
        regs.x.ax = 0x000a;                       /* Funct. no. for "Set text pointer type" */
        regs.x.bx = 0;                            /* Create software pointer */
        regs.x.cx = mask;                         /* Low word is AND-mask */
        regs.x.dx = mask >> 16;                   /* High word is XOR-mask */
        MOUINT(regs, regs);                       /* Call mouse driver */
        oldercursor = mask;                       /* Note old bitmask */
    }
}

/*****
* Function          : M o u E v e n t H a n d l e r
*-----
* Task              : Calls AssmHand routine from mouse driver, when
*                   : a mouse related event occurs.
* Input parameters : EvFlags = Event's event mask
*                   : ButState = Mouse button status
*                   : X, Y    = Current pointer position, converted
*                   : into text screen coordinates
* Return value      : None
* Info              : - This function is only operational through a
*                   : mouse driver call, and shouldn't be called
*                   : from another function.
*****
void MouEventHandler( int EvFlags, int ButState, int x, int y )
{
    #define LBITS ( EV_LEFT_PRESS | EV_LEFT_REL )
    #define RBITS ( EV_RIGHT_PRESS | EV_RIGHT_REL )

    unsigned newrng;                                /* New range number */

    mouevent &= ~1;                                /* Clear bit 0 */
    mouevent |= ( EvFlags & 1 );                    /* Copy EvFlags to bit 0 */

    if ( EvFlags & LBITS )                          /* Left mouse button pressed or released? */
    {                                               /* YES */
        mouevent &= ~LBITS;                       /* Clear previous status */
        mouevent |= ( EvFlags & LBITS );           /* Add new status */
    }

    if ( EvFlags & RBITS )                          /* Right mouse button pressed or released? */
    {                                               /* YES, Clear and set bits */
        mouevent &= ~RBITS;                       /* Clear previous status */
        mouevent |= ( EvFlags & RBITS );           /* Add new status */
    }

    moucol = x;                                     /* Convert columns into text columns */
    mourow = y;                                     /* Convert rows into text rows */

    /*-- Check range in which mouse is currently located, and compare --*/
    /*-- to range since last call. If a change occurs, the pointer's ---*/
    /*-- appearance will have to be changed. ---*/

    newrng = *(bbuf + mourow * tcol + moucol);     /* Get range */
    if ( newrng != mourng )                         /* New range? */
        MouDefinePtr((newrng==NO_RANGE) ? stdptr :
                      (cur_range+newrng)->ptr_mask);
    mourng = newrng;                               /* Place range number in global variables */
}

#pragma check_stack                                     /* Re-enable stack checking and old */

```

```

#pragma check_stack                /* status */

/*****
* Function      : M o u I B u f F i l l
**-----**
* Task          : Stores a specific screen range code within
*                screen memory affecting the module
* Input parameters : x1, y1 = Upper left corner of the screen
*                  x2, y2 = Lower right corner of the screen
*                  CODE   = Range code
* Return value   : None
* Info           : This functions should only be called from within
*                this module.
*****/

static void MouIBufFill( BYTE x1, BYTE y1,
                        BYTE x2, BYTE y2, BYTE code )
{
    register BYTE * lptr;           /* Floating pointer to range mem. */
    BYTE i, j;                     /* Loop counter */

    lptr = bbuf + y1 * tcol + x1;  /* Pointer to first line */

    /*-- Go through individual lines -----*/
    for (j=x2 - x1 + 1 ; y1 <= y2; ++y1, lptr+=tcol )
        memset( lptr, code, j );   /* Set code */
}

/*****
* Function      : M o u D e f R a n g e
**-----**
* Task          : Allows the definition of different screen ranges
*                which configure a different code for the mouse
*                pointer, depending on the pointer's location.
* Input parameters : - NUMBER = Number of screen ranges
*                  - PTR     = Pointer to screen description vector
*                        (type RANGE)
* Return value   : None
* Info           : - Free screen ranges receive the code NO_RANGE.
*                  - When entering the specified screen range, the
*                    mouse handler automatically changes the mouse
*                    pointer's appearance to correspond with that
*                    range.
*                  - Since the specified pointer is stored, but the
*                    specified vector isn't copied to a separate
*                    buffer, the contents of the vector should not
*                    be changed on the next call of this function.
*****/

void MouDefRange( BYTE number, RANGE * ptr )
{
    register BYTE i,               /* Loop counter */
                 range;           /* Mouse range */

    cur_range = ptr;               /* Reserve pointer to vector */
    num_range = number;           /* and number of ranges */
    memset( bbuf, NO_RANGE, blen );
    for (i=0 ; i<number ; ++ptr )
        MouIBufFill( ptr->x1, ptr->y1, ptr->x2, ptr->y2, i++);

    /*-- Redefine mouse pointer -----*/

    range = *(bbuf + mourow * tcol + moucol); /* Current mouse range */
    MouDefinePtr( ( range == NO_RANGE ) ? stdptr
                 : (cur_range+range)->ptr_mask );
}

/*****
* Function      : M o u E v e n t W a i t
**-----**
* Task          : Waits for a specific event from the keyboard.
* Input parameters : TYP          = Establishes comparison between
*                        different events.
*                  WAIT_EVENT = Bitmask which specifies wait event.
* Return value   : Bitmask which describes this or another event.
* Info           : - WAIT_EVENT can be used with other constants
*****/

```

```

*           such as EV_MOUSE_MOVE or EV_LEFT_PRESS when used *
*           in conjunction with EVOR.                          *
*           - EAND & EVOR are allowable types. EAND has the *
*           ability to return to the caller once ALL events*
*           have occurred; EVOR returns to the caller when *
*           at least one event occurs.                        *
*****/

```

```

int MouEventWait( BYTE typ, int wait_event )
{
    int cur_event;                /* Current event mask */
    register BYTE column = moucol, /* Last mouse position */
               line = mourow;
    BYTE ende = FALSE;           /* TRUE if an event occurs */

    while ( !ende )              /* Repeat until event occurs */
    {
        /*-- Wait until one of the events occurs -----*/

        if ( typ == EAND )        /* EAND: All events must occur */
            while ( (cur_event = mouevent) != wait_event )
                ;
        else                      /* EVOR: At least one event must occur */
            while ( ( (cur_event = mouevent) & wait_event) == 0 )
                ;

        cur_event &= wait_event;  /* Check event bits only */

        /*-- When moving the mouse, the event is only accepted if the --*/
        /*-- pointer moves to another row or column on the text screen --*/

        if ((wait_event & EV_MOUSE_MOVE) && column==moucol && line==mourow)
        {
            /* Mouse moves, but in same screen position */
            cur_event &= (~EV_MOUSE_MOVE); /* Examine move bit */
            ende = (cur_event != 0);        /* Are events pending? */
        }
        else                      /* Event occurred */
            ende = TRUE;
    }
    ev_col = moucol;              /* Set current mouse position */
    ev_row = mourow;              /* and mouse range; place in */
    ev_rng = mourng;              /* global variables */
    return( cur_event );          /* Return event mask */
}

```

```

/*****
* Function      : M o u I S e t E v e n t H a n d l e r      *
**-----**
* Task          : Installs an event handler which handles events *
*                called from the mouse driver.                *
* Input parameters : EVENT = Bitmask which specifies the event which *
*                calls the event handler.                      *
*                PTR   = Pointer to the mouse handler          *
* Return value   : None                                         *
* Info          : - EVENT can be used in conjunction with the EVOR *
*                comparison on constants such as EV_MOUSE_MOVE, *
*                EV_LEFT_PRESS                                  *
*****/

```

```

static void MouISetEventHandler( unsigned event, MOUHAPTR ptr )
{
    union REGS regs;              /* Processor regs for interrupt call */
    struct SREGS sregs;          /* Segment register for interrupt call */

    regs.x.ax = 0x000C;           /* Funct. no. for "Set Mouse Handler" */
    regs.x.cx = event;            /* Load event mask */
    regs.x.dx = FP_OFF( ptr );    /* Offset address of handler */
    sregs.es = FP_SEG( ptr );     /* Segment address of handler */
    MOUINTX( regs, regs, sregs ); /* Call mouse driver */
}

```

```

/*****
* Function      : M o u I G e t X                          *
**-----**
* Task          : Determines text column in which pointer lies. *
* Input parameters : None                                         *

```

```

* Return value      : Mouse pointer column, relative to text screen *
*****/

static BYTE MouGetX( void )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax= 0x0003;              /* Funct. no. for "Get mouse position" */
    MOUINT( regs, regs );           /* Call mouse driver */
    return XTocol( regs.x.cx );     /* Convert and return column */
}

/*****
* Function          : M o u I G e t Y *
**-----**
* Task              : Determines text row in which pointer lies. *
* Input parameters  : None *
* Return value      : Mouse pointer row, relative to the text screen *
*****/

static BYTE MouGetY( void )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax= 0x0003;              /* Funct. no. for "Get mouse position" */
    MOUINT( regs, regs );           /* Call mouse driver */
    return YTORow( regs.x.dx );     /* Convert and return row */
}

/*****
* Function          : M o u S h o w M o u s e *
**-----**
* Task              : Display mouse pointer on the screen. *
* Input parameters  : None *
* Return value      : None *
* Info              : Calls of MouHidemMouse() and MouShowMouse() must *
                    : be kept balanced. *
*****/

void MouShowMouse( void )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax = 0x0001;             /* Funct. no. for "Show Mouse" */
    MOUINT( regs, regs );           /* Call mouse driver */
}

/*****
* Function          : M o u H i d e M o u s e *
**-----**
* Task              : Hide mouse pointer from screen. *
* Input parameters  : None *
* Return value      : None *
* Info              : Calls of MouHidemMouse() and MouShowMouse() must *
                    : be kept balanced. *
*****/

void MouHideMouse( void )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax = 0x0002;             /* Funct. no. for "Hide Mouse" */
    MOUINT( regs, regs );           /* Call mouse driver */
}

/*****
* Function          : M o u S e t M o v e A r e a *
**-----**
* Task              : Defines a screen range within which the mouse *
                    : pointer may be moved. *
* Input parameters  : x1, y1 = Coordinates of upper left corner *
                    : x2, y2 = Coordinates of lower right corner *
* Return value      : None *
* Info              : - Both parameters apply to text screen, NOT the *
                    : mouse driver's virtual graphic screen - *
*****/

```

```

void MouSetMoveArea( BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax = 0x0008;              /* Funct. no. for "Set vertical Limits" */
    regs.x.cx = ROWTOY( y1 );        /* Conversion to virtual */
    regs.x.dx = ROWTOY( y2 );        /* mouse screen */
    MOUINT(regs, regs);              /* Call mouse driver */
    regs.x.ax = 0x0007;              /* Funct. no. for "Set horizontal Limits" */
    regs.x.cx = COLTOX( x1 );        /* Conversion to virtual */
    regs.x.dx = COLTOX( x2 );        /* mouse screen */
    MOUINT(regs, regs);              /* Call mouse driver */
}

```

```

/*****
* Function      : M o u S e t S p e e d
**-----**
* Task          : Determines the difference between mouse movement
*                speed and the resulting pointer speed on the
*                screen.
* Input parameters : - XSPEED = Horizontal speed
*                  - YSPEED = Vertical speed
* Return value   : None
* Info          : - Both parameters are based on mickeys
*                (mickey / 8 pixel).
*****/

```

```

void MouSetSpeed( int xspeed, int yspeed )
{
    union REGS regs;                /* Processor regs for interrupt call */

    regs.x.ax = 0x000f;             /* Funct. no. for "Set mickeys to pixel ratio" */
    regs.x.cx = xspeed;
    regs.x.dx = yspeed;
    MOUINT(regs, regs);             /* Call mouse driver */
}

```

```

/*****
* Function      : M o u M o v e P t r
**-----**
* Task          : Moves the mouse pointer to a specific position
*                on the screen.
* Input parameters : - COL = new screen column
*                  - ROW = new screen row
* Return value   : None
* Info          : - Both parameters apply to the text screen, NOT
*                to the mouse driver's virtual graphic screen
*****/

```

```

void MouMovePtr( int col, int row )
{
    union REGS regs;                /* Processor regs for interrupt call */
    unsigned newrng;               /* Range in which the mouse can move */

    regs.x.ax = 0x0004;             /* Funct. no. for "Set mouse pointer position" */
    regs.x.cx = COLTOX( moucol = col ); /* Convert coordinates and store */
    regs.x.dx = ROWTOY( mourow = row ); /* in global variables */
    MOUINT(regs, regs);             /* Call mouse driver */

    newrng = *(bbuf + mourow * tcol + moucol); /* Get range */
    if ( newrng != mourng )          /* New range? */
        MouDefinePtr((newrng==NO_RANGE) ? stdptr :
                     (cur_range+newrng)->ptr_mask);
    mourng = newrng;                /* Place range number in global variables */
}

```

```

/*****
* Function      : M o u S e t D e f a u l t P t r
**-----**
* Task          : Defines mouse pointer for screen ranges without
*                the help of MouDefRange.
* Input parameters : STANDARD = Bitmask for standard mouse pointer
* Return value   : None
*****/

```

```

void MouSetDefaultPtr( PTRVIEW standard )
{
    stdptr = standard;                /* Place bitmask in global variables */

    /*-- If mouse is currently in no range, go direct to conversion ---*/
    /*-- to new pointer appearance ---*/

    if ( MouGetRange() == NO_RANGE )    /* Not in any range? */
        MouDefinePtr( standard );      /* NO */
}

/*****
* Function      : M o u E n d
**-----**
* Task         : Ends mouseC module functions.
* Input parameters : None
* Return value  : None
* Info        : Function is called automatically when program
*              ends, as long as MouInstall is called first.
*****/

void MouEnd( void )
{
    union REGS regs;                /* Processor regs for interrupt call */

    MouHideMouse();                /* Hide mouse pointer from screen */
    regs.x.ax = 0;                  /* Reset mouse driver */
    MOUINT(regs, regs);             /* Call mouse driver */

    free( bbuf );                  /* Release allocated memory */
}

/*****
* Function      : M o u I n i t
**-----**
* Task         : Initializes variables and mouseC module
* Input parameters : Columns, = Text screen resolution
*                Lines
* Return value  : TRUE if a mouse is installed, else FALSE
* Info        : This function must be called as the first one in
*              the module.
*****/

BYTE MouInit( BYTE columns, BYTE lines )
{
    union REGS regs;                /* Processor regs for interrupt call */

    tline = lines;                 /* Store no. of lines and cols */
    tcol = columns;                /* in global variables */

    atexit( MouEnd );              /* Call MouEnd at end of program */

    /*-- Allocate and fill mouse range buffer -----*/

    bbuf = (BYTE *) malloc( blen = tline * tcol );
    MouIBufFill( 0, 0, tcol-1, tline-1, NO_RANGE );

    regs.x.ax = 0;                 /* Initialize mouse driver */
    MOUINT(regs, regs);             /* Call mouse driver */
    if ( regs.x.ax != 0xffff )      /* Mouse driver installed? */
        return FALSE;              /* NO */

    MouSetMoveAreaAll();           /* Set range of movement */

    moucol = MouIGetX();            /* Load current mouse pos. */
    mourow = MouIGetY();           /* into global variables */

    /*-- Install assembler event handler "AssmHand" -----*/
    MouISetEventHandler( EV_MOU_ALL, (MOUHAPTR) AssmHand );

    return mavail = TRUE;          /* Mouse is installed */
}

/*****
* M A I N P R O G R A M
*****/

```



```

int main( void )
{
    static RANGE ranges[] =                                /* Mouse ranges */
    {
        { 0, 0, 79, 0, MouPtrMask( PTRDIFCHAR(0x18), PTRINVCOL) },
        { 0, 1, 0, 23, MouPtrMask( PTRDIFCHAR(0x1b), PTRINVCOL) },
        { 0, 24, 78, 24, MouPtrMask( PTRDIFCHAR(0x19), PTRINVCOL) },
        { 79, 1, 79, 23, MouPtrMask( PTRDIFCHAR(0x1a), PTRINVCOL) },
        { 79, 24, 79, 24, MouPtrMask( PTRDIFCHAR('X'), PTRDIFCOLB(0x40) ) },
    };

    printf("\nMOUSEC - (c) 1989 by MICHAEL TISCHER\n\n");
    if ( MouInit( 80, 25 ) )                                /* Initialize mouse module */
    {
        /* OK, there is an installed mouse driver */
        printf("Move the mouse pointer around on the screen. When you move\n"
            "the mouse pointer to the border of the screen, the\n"
            "mouse pointer changes in appearance, depending upon its\n"
            "Current position.\n\n"
            "Move the mouse pointer to the lower right corner of the\n"
            "screen, and press both the left and right mouse buttons\n"
            "to end this demo program.\n" );

        MouSetDefaultPtr( MouPtrMask( PTRDIFCHAR( '[' ), PTRDIFCOL( 3 ) ) );
        MouDefRange( ELVEC( ranges ), ranges );              /* Range definition */
        MouShowMouse();                                     /* Display mouse pointer on the screen */

        /*-- Wait until the user presses the left and right mouse      --*/
        /*-- buttons simultaneously, AND the mouse pointer lies in      --*/
        /*-- range 4                                                      --*/

        do                                                    /* Read loop */
            MouEventWait( EAND, EV_LEFT_PRESS | EV_RIGHT_PRESS );
        while ( MouGetRange() != 4 );

        return 0;                                            /* Return OK code to DOS */
    }
    else                                                    /* No mouse OR mouse driver installed */
    {
        printf("Sorry, no mouse driver installed.\n");
        return 1;                                           /* Return error code to DOS */
    }
}

```