

```

/*****
/*          M E M D E M O C . C          */
/*-----*/
/*   Task           : Demonstrates DOS memory management.   */
/*-----*/
/*   Memory model    : SMALL                                */
/*-----*/
/*   Author          : Michael Tischer                      */
/*   Developed on     : 10/08/91                             */
/*   Last update      : 04/07/95                             */
*****/

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

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

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

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

/*-- Function numbers for interrupt 0x21 -----*/

#define GET_MEM      0x48          /* Reserve RAM */
#define FREE_MEM     0x49          /* Release RAM */
#define CHANGE_MEM   0x4A          /* Change size of a memory range */
#define GET_STRATEGY 0x5800        /* Get memory division strategy */
#define SET_STRATEGY 0x5801        /* Set memory division strategy */
#define GET_UMB      0x5802        /* Get UMB */
#define SET_UMB      0x5803        /* Set UMB */

/*-- Search strategies for SET_STRATEGY -----*/

#define SRCHS_BELOW  0x00          /* Use the first free memory block */
#define SRCHS_BEST   0x01          /* Use the best memory block */
#define SRCHS_ABOVE  0x02          /* Use the last free memory block */
#define SFIRST_UMB   0x80          /* Search for first in UMB */
/* (combined with SEARCH_...) */

/*-- Constants for SetUMB -----*/

#define UMB_OUT       0x00          /* Ignore UMB */
#define UMB_INC       0x01          /* Include UMB in memory allocation */

/*-- Constants for Demo -----*/

#define TEST_EN       (10240-1)    /* 10239 paragraph test environment */
#define TEST_EN_UMB   (2560-1)/* 2559 paragraph UMB test environment */
#define TEST_EN_K      160          /* Test environment is 160K */
#define TEST_EN_UMB_K  40           /* Test environment is 40K */
#define BLOCKAMT      26           /* Number of addresses for result display */

/*-- Key codes for control -----*/

#define ESC           27            /* Press <Esc> to cancel */
#define F1            59            /* Function key F1 */
#define F2            60            /* Function key F2 */
#define F3            61            /* Function key F3 */
#define F8            66            /* Function key F8 */
#define F9            67            /* Function key F9 */
#define F10           68            /* Function key F10 */

/*== Type declarations =====*/

typedef struct { unsigned int SegAddr,          /* Segment address */
                MBlSize;
                } BlockType;                /* Memory size */

typedef unsigned char BYTE;

```

```

/*== Macros =====*/

#ifdef MK_FP                                /* Macro MK_FP already defined? */
    #undef MK_FP                            /* Yes --> Undefine macro */
#endif

#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))

/*== Typed constants =====*/

char *YesNo[]      = { "No ", "Yes " };
char *SText[]      = { "First free memory block used",
    "Best free memory block used ",
    "Last free memory block used " };
BYTE ColArray[]    = { 0x07, 0x70 };
char *Keybd_Text[] = { " [F1] Allocate memory",
    " [F2] Release memory",
    " [F3] Change size",
    " [ESC] End program" };

/*== Global variables =====*/

union REGS  regs;                /* Registers for interrupt call */
struct SREGS sregs;             /* Segment registers for extended interrupt */
unsigned int MAddrArray[ 1000 ]; /* Array for memory */
unsigned int Num_Addrs;          /* Number of addresses */
unsigned int ConvSeg;            /* Address of test block in conventional RAM */
unsigned int UMBSeg;             /* Address of test block in UMB */
BlockType   BlocArray[ BLOCKAMT ]; /* Memory */

/*== Screen routines for Microsoft C =====*/

#ifdef __TURBOC__                /* Microsoft C? */

#define textcolor( Color )
#define textbackground( Color )

/*****
/* Gotoxy      : Places the cursor.
/* Input       : Cursor coordinates
/* Output      : None
*****/

void gotoxy( int x, int y )
{
    regs.h.ah = 0x02;            /* Function number for interrupt call */
    regs.h.bh = 0;               /* Color */
    regs.h.dh = y - 1;
    regs.h.dl = x - 1;
    int86( 0x10, &regs, &regs ); /* Interrupt call */
}

/*****
/* clrscr      : Clears the screen.
/* Input       : See below
/* Output      : None
*****/

void clrscr( void )
{
    regs.h.ah = 0x07;            /* Function number for interrupt call */
    regs.h.al = 0x00;
    regs.h.ch = 0;
    regs.h.cl = 0;
    regs.h.dh = 24;
    regs.h.dl = 79;
    int86( 0x10, &regs, &regs ); /* Interrupt call */
    gotoxy( 1, 1 );             /* Set cursor */
}

#endif

/*****
/* PRINT : Similar to PRINTF, writes the string directly to
/*       video RAM.
*****/

```

```

/* Input      : COLUMN = Display column          */
/*            SCROW  = Display row              */
/*            DCOLR  = Display color            */
/*            STRING = Pointer to PRINTF string  */
/*            ...    = other arguments          */
/* Output     : None                            */
/*****

```

```

void Print( int Column, int ScRow, BYTE DColr, char * String, ...)
{
    struct vr {
        BYTE DisChar,          /* 2 bytes to a screen position */
        Attribute;             /* ASCII code */
    } far * lptr;             /* Corresponding attribute */
    va_list parameter;         /* Floating pointer to video RAM access */
    char Output[255];          /* Parameter list for VA... macros */
    *aptr = Output;           /* Buffer for formatted string */
    static unsigned int viosege = 0; /* For string execution */
    union REGS Register;       /* Register variables for interrupt call */

    if ( viosege == 0 )        /* First call? */
    {
        /* Yes --> Get segment address of video RAM */
        Register.h.ah = 0x0F;
        int86(0x10, &Register, &Register);
        viosege = ( Register.h.al == 7 ? 0xb000 : 0xb800 );
    }

    va_start( parameter, String ); /* Convert parameter */
    vsprintf( Output, String, parameter ); /* Format */
    lptr = (struct vr far *)
        MK_FP( viosege, ( (ScRow-1) * 80 + (Column-1) ) << 1 );

    for ( ; *aptr ; )          /* Run string */
    {
        lptr->DisChar = *aptr++; /* Character in video RAM */
        lptr++->Attribute = DColr; /* Set character attribute */
        ++Column;
    }
    gotoxy( Column, ScRow );    /* Place cursor after output */
}

```

```

/*****
/* DOS_GetMem : Reserves memory.                */
/* Input      : Desired memory size in paragraphs. */
/* Output     : Segment address of the allocated memory block, */
/*            number of paragraphs allocated or */
/*            maximum number of available paragraphs */
/*****

```

```

void DOS_GetMem( unsigned int Ps,
                unsigned int *Adr,
                unsigned int *Res )
{
    regs.h.ah = GET_MEM; /* Function number */
    regs.x.bx = Ps;       /* Number of paragraphs to be reserved */
    intdos( &regs, &regs ); /* Interrupt call */
    if ( !regs.x.cflag )    /* Call successful? */
    {
        *Adr = regs.x.ax; /* Yes --> Return address and size */
        *Res = Ps;
    }
    else /* No --> Error */
    {
        *Adr = 0; /* No memory reserved */
        *Res = regs.x.bx; /* Maximum available size */
    }
}

```

```

/*****
/* DOS_FreeMem: Releases reserved memory.        */
/* Input      : Memory segment address            */
/* Output     : None                              */
/*****

```

```

void DOS_FreeMem( unsigned int Adr )
{

```

```

regs.h.ah = FREE_MEM;                                /* Function number */
sregs.es = Adr;                                       /* Address of memory to be released */
intdosx( &regs, &regs, &sregs );                     /* Interrupt call */
}

/*****
/* DOS_ChangeMem : Changes reserved memory size.
/* Input      : Old segment address, new size in paragraphs
/* Output     : Segment address of the allocated memory block,
/*            : number of paragraphs allocated or
/*            : maximum number of available paragraphs
*****/

void DOS_ChangeMem( unsigned int Ps,
                   unsigned int *Adr,
                   unsigned int *Res )
{
    regs.h.ah = CHANGE_MEM;                          /* Function number */
    regs.x.bx = Ps;                                   /* Number of paragraphs to be reserved */
    sregs.es = *Adr;                                  /* Segment address of memory block to be changed */
    intdosx( &regs, &regs, &sregs );                 /* Interrupt call */
    if ( !regs.x.cflag )                             /* Call successful? */
        *Res = Ps;                                   /* Yes --> New memory size */
    else                                              /* No --> Error */
        *Res = regs.x.bx;                            /* Maximum available memory size */
}

/*****
/* ReadStrategy : Reads the DOS memory management strategy.
/* Input      : None
/* Output     : Memory management strategy
*****/

int ReadStrategy( void )
{
    regs.x.ax = GET_STRATEGY;                        /* Set function number */
    intdos( &regs, &regs );
    return regs.x.ax;                                /* Display strategy */
}

/*****
/* ReadUMB      : Reads UMB status.
/* Input      : None
/* Output     : UMB ignored during memory management
/* Info       : First available in DOS Version 5.0.
*****/

int ReadUMB( void )
{
    regs.x.ax = GET_UMB;                             /* Set function number */
    intdos( &regs, &regs );                         /* Interrupt call */
    return regs.h.al;                                /* Display status */
}

/*****
/* SetDosStrategy: Sets DOS memory management strategy.
/* Input      : New strategy
/* Output     : None
*****/

void SetDosStrategy( unsigned int Strategy )
{
    regs.x.ax = SET_STRATEGY;                        /* Set function number */
    regs.x.bx = Strategy;
    intdos( &regs, &regs );
}

/*****
/* SetUMB      : Sets the UMB status.
/* Input      : New UMB status
/* Output     : None
/* Info       : First available in DOS Version 5.0.
*****/

void SetUMB( unsigned int UMB )
{

```

```

regs.x.ax = SET_UMB; /* Set function number */
regs.x.bx = UMB;
intdos( &regs, &regs );
}

/*****
/* Allocate_Memory : Creates a test environment for memory test.
/* Input : None
/* Output : None
*****/

void Allocate_Memory( void )
{
    unsigned int SegAdr; /* Segment address of allocated memory */
    unsigned int Des_Mem; /* Desired memory size */
    unsigned int MBlSize; /* Size of allocated memory range */

    /*-- 1. Allocate test block -----*/

    SetUMB( UMB_OUT );
    DOS_GetMem( TEST_EN, &ConvSeg, &MBlSize ); /* Get test block */
    if ( ConvSeg == 0 ) /* Error? */
        return; /* Yes --> Return to caller */

    SetUMB( UMB_INC );
    SetDosStrategy( SRCHS_ABOVE | SFIRST_UMB );
    DOS_GetMem( TEST_EN_UMB, &UMBSeg, &MBlSize );
    if ( UMBSeg != 0 && UMBSeg < 0xA000 ) /* No UMB available? */
    {
        DOS_FreeMem( UMBSeg ); /* Release memory */
        UMBSeg = 0; /* No UMBS */
    }

    /*-- 2. Allocate remaining conventional/ UMB memory in 1K blocks ---*/

    Des_Mem = 63; /* First try allocating 15 paragraphs */
    Num_Addrs = 0;
    do
    {
        DOS_GetMem( Des_Mem, &SegAdr, &MBlSize ); /* Call for memory */
        if ( SegAdr != 0 ) /* Memory received? */
            MAddrArray[ Num_Addrs++ ] = SegAdr; /* Yes --> Note address */
    }
    while ( SegAdr != 0 ); /* Allocate all memory */

    /*-- 3. Release test blocks -----*/

    if ( ConvSeg > 0 )
        DOS_FreeMem( ConvSeg-- ); /* MCB is also free */
    if ( UMBSeg > 0 )
        DOS_FreeMem( UMBSeg-- ); /* MCB is also free */
}

/*****
/* ReleaseThisMemory: Releases reserved memory.
/* Input : None
/* Output : None
/* Global variables : MAddrArray/R
*****/

void ReleaseThisMemory( void )
{
    unsigned int i; /* Loop counter */

    if ( Num_Addrs > 0 ) /* Release individual 1K blocks */
        for ( i = 0; i < Num_Addrs; i++ )
            DOS_FreeMem( MAddrArray[ i ] );
}

/*****
/* DisplayMemLayout : Displays memory layout.
/* Input : W_Borders = TRUE if the border should also
/* be displayed
/* Output : None
*****/

```

```

void DisplayMemLayout( BYTE W_Borders )
{
    char          SArray[ TEST_EN_K ];
    char          SArray_UMB[ TEST_EN_UMB_K ];
    unsigned int  i, j;
    unsigned int  Position;
    char          Sdummy[ 20 ];
    char          LastChar_M;
    int           CurColor;

    /* Loop counter */
    /* Help variable */

    /* Memory for last character displayed */
    /* Current output color */

    memset( SArray, 32, TEST_EN_K );
    memset( SArray_UMB, 32, TEST_EN_UMB_K );

    /*-- Fill memory table -----*/

    for ( i = 0; i < BLOCKAMT; i++ )
    {
        if ( BlocArray[ i ].SegAddr > 0xA000 )
        {
            /* UMB? */
            Position = ( BlocArray[ i ].SegAddr - UMBSeg ) / 64;
            for ( j = 0; j <= BlocArray[ i ].MBLSize / 64; j++ )
                SArray_UMB[ Position + j ] = i + 65;
        }
        else if ( BlocArray[ i ].SegAddr > 0 )
        {
            Position = ( BlocArray[ i ].SegAddr - ConvSeg ) / 64;
            for ( j = 0; j <= BlocArray[ i ].MBLSize / 64; j++ )
                SArray[ Position + j ] = i + 65;
        }
    }

    /*-- Draw border around memory table -----*/

    if ( W_Borders )
    {
        Print( 1, 7, 0x07, "Conventional memory:" );
        Print( 1, 8, 0x07,
            "
            1          2          3          4" );
        Print( 1, 9, 0x07,
            "
            1          0          0          0          0" );
        Print( 1, 10, 0x07,
            "+-----+" );
        for ( i = 0; i < 4; i++ )
            Print( 1, 11 + i, 0x07, "| %3i %s%s", i * 40,
                "K |",
                Keybd_Text[ i ] );
        Print( 1, 15, 0x07,
            "+-----+" );
        if ( UMBSeg > 0 )
        {
            Print( 1, 17, 0x07, "UMB:" );
            Print( 1, 18, 0x07,
                "
                1          2          3          4" );
            Print( 1, 19, 0x07,
                "
                1          0          0          0          0" );
            Print( 1, 20, 0x07,
                "+-----+" );
            Print( 1, 21, 0x07,
                "| 0 KB |" );
            Print( 1, 22, 0x07,
                "+-----+" );
        }
        else
            Print( 1, 17, 0x07, "No UMB available" );
    }

    /*-- Display table for conventional memory -----*/

    LastChar_M = 0;
    CurColor = 1;
    /* Last character displayed */
    /* Last display color */
    for ( i = 0; i < 4; i++ )
        for ( j = 0; j < 40; j++ )
        {
            if ( LastChar_M != SArray[ i * 40 + j ] )
                /* Color change? */

```

```

    {
        CurColor = ( CurColor + 1 ) % 2;          /* New color number */
        LastChar_M = SArray[ i * 40 + j ];        /* Chars. to compare */
    }
    Print( j + 11, i + 11, ColArray[ CurColor ],
           "%c", SArray[ i * 40 + j ] );
}

/*-- Display table for UMB -----*/

if ( UMBSeg > 0 )
{
    for ( j = 0; j < 40; j++ )
    {
        if ( LastChar_M != SArray_UMB[ j ] )      /* Color change? */
        {
            CurColor = ( CurColor + 1 ) % 2;      /* New color number */
            LastChar_M = SArray_UMB[ j ];          /* Chars. to compare */
        }
        Print( j + 11, 21, ColArray[ CurColor ],
               "%c", SArray_UMB[ j ] );
    }
}

/*****
/* Demo          : Demonstrates memory management.          */
/* Input         : Included UMB, first UMB to search,        */
/*               : memory division strategy                  */
/* Output        : None                                       */
*****/

void Demo( int Inc_UMB,
           int UMB_first,
           int Strategy )
{
    int i;                                /* Loop counter */
    int IKeys;                            /* Input keys */
    char ResPtr[ 5 ]; /* Pointer to desired reservation (A-Z) */
    unsigned int Des_Mem; /* Size of reservation */
    unsigned int MB1Size;
    char *sdummy[ 20 ];

    /*-- Initialize address array -----*/

    for ( i = 0; i < BLOCKAMT; i++ )      /* Initialize all blocks */
    {
        BlocArray[ i ].SegAddr = 0;        /* Segment address of memory block */
        BlocArray[ i ].MB1Size = 0;        /* Size of memory block */
    }

    DisplayMemLayout( TRUE );              /* Display memory table */

    /*-- Demonstration loop -----*/

    do
    {
        /*-- Specify selected memory management strategy -----*/

        if ( Inc_UMB ) /* UMB used? */
            SetUMB( UMB_INC );
        else
            SetUMB( UMB_OUT );

        if ( UMB_first )
            SetDosStrategy( Strategy | SFIRST_UMB );
        else
            SetDosStrategy( Strategy );

        /*-- Display current strategy -----*/

        Print( 1, 3, 0x07,
               "Memory management strategy: %s [F8]",
               SText[ Strategy ] );
        Print( 1, 4, 0x07,
               "First search in UMB : %s %s",

```

```

        YesNo[ UMB_first ], "                                [F9]" );
Print( 1, 5, 0x07,
        "UMB used                                : %s          %s",
        YesNo[ Inc_UMB ], "                                [F10]" );
Print( 1, 6, 0x07, "-----" );
Print( 40, 6, 0x07, "-----" );

/*-- Entry and processing -----*/

while ( ! kbhit() );                                /* Wait for a key */
IKeys = getch();                                    /* Read key */
if ( ( IKeys == 0 ) && ( kbhit() ) )                /* Function key */
    IKeys = getch();                                /* Get 2nd key code */

switch( IKeys )
{
case F1 :
    /* Allocate memory block */
    i = -1;                                         /* No more valid memory blocks given */
    do
    {
        /* Enter up to valid block */
        Print( 1, 23, 0x07, "Reserve which block [ A-Z ]: " );
        scanf( "%s", ResPtr );
        ResPtr[0] = toupper( ResPtr[ 0 ] );
        if ( ( ResPtr[0] >= 'A' ) && ( ResPtr[0] <= 'Z' ) )
            if ( BlocArray[ (int) ResPtr[0] - 65 ].SegAddr == 0 )
                i = (int) ResPtr[0] - 65;
    }
    while ( i == -1 );
    Print( 1, 24, 0x07,
        "How many K do you want reserved: " );
    scanf( "%i", &Des_Mem );
    Des_Mem = Des_Mem * 64 - 1;                    /* Convert to paragraphs */
    DOS_GetMem( Des_Mem, &BlocArray[ i ].SegAddr,
        &BlocArray[ i ].MBLSize );
    if ( BlocArray[ i ].MBLSize != Des_Mem )        /* Error? */
    {
        Print( 1, 25, 0x07, "Only %4dK free",
            ( BlocArray[ i ].MBLSize + 1 ) / 64 );
        while ( !kbhit() )
            ;
        while ( kbhit() )
            IKeys = getch();
        IKeys = 0;
    }
    Print(1, 23, 0x07, "                                ");
    Print(1, 24, 0x07, "                                ");
    Print(1, 25, 0x07, "                                ");
    DisplayMemLayout( FALSE );                      /* Display memory table */
    break;

case F2 :
    /* Release memory block */
    i = -1;                                         /* No more valid blocks given */
    do
    {
        /* Enter up to valid block */
        Print( 1, 23, 0x07, "Release which block [ A-Z ]: " );
        scanf( "%s", ResPtr );
        ResPtr[0] = toupper( ResPtr[ 0 ] );
        if ( ( ResPtr[0] >= 'A' ) && ( ResPtr[0] <= 'Z' ) )
            if ( BlocArray[ (int) ResPtr[0] - 65 ].SegAddr != 0 )
                i = (int) ResPtr[0] - 65;
    }
    while ( i == -1 );
    DOS_FreeMem( BlocArray[ i ].SegAddr );
    BlocArray[ i ].SegAddr = 0;
    BlocArray[ i ].MBLSize = 0;
    Print(1, 23, 0x07, "                                ");
    DisplayMemLayout( FALSE );                      /* Display memory table */
    break;

case F3 :
    /* Change one block's size */
    i = -1;                                         /* No more valid blocks given */
    do
    {
        /* Enter up to valid block */
        Print( 1, 23, 0x07, "Which block do you want changed [ A-Z ]: " );
        scanf( "%s", ResPtr );
        ResPtr[0] = toupper( ResPtr[ 0 ] );

```



```

        if ( ( ResPtr[0] >= 'A' ) && ( ResPtr[0] <= 'Z' ) )
            if ( BlocArray[ (int) ResPtr[0] - 65 ].SegAddr != 0 )
                i = (int) ResPtr[0] - 65;
    }
    while ( i == -1 );
    Print( 1, 24, 0x07, "How many K do you want reserved: " );
    scanf( "%i", &Des_Mem );
    Des_Mem = Des_Mem * 64 - 1; /* Convert to paragraphs */
    DOS_ChangeMem( Des_Mem, &BlocArray[ i ].SegAddr, &MBLSize );
    if ( MBLSize != Des_Mem ) /* Error? */
    {
        Print( 1, 23, 0x07, "Only %4iK free",
                ( MBLSize + 1 ) / 64 );
        while ( !kbhit() );
        while ( kbhit() )
            IKeys = getch();
        IKeys = 0;
    }
    else
        BlocArray[ i ].MBLSize = MBLSize; /* Set new size */
    Print(1, 23, 0x07, " ");
    Print(1, 24, 0x07, " ");
    Print(1, 25, 0x07, " ");
    DisplayMemLayout( FALSE ); /* Display memory table */
    break;

case F8 : /* Change strategy */
    Strategy = ( Strategy + 1 ) % 3;
    break;

case F9 : /* Toggle: UMB first */
    UMB_first = ! UMB_first;
    break;

case F10: /* Toggle: Add UMB */
    Inc_UMB = ! Inc_UMB;
    break;
}
}
while ( IKeys != ESC );
}

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

void main( void )
{
    int StartStrategy; /* Memory division on startup */
    int StartUMB;      /* UMB layout on startup */
    int Cur_UMB_INC;    /* UMB used (yes/no) */
    int Cur_UMB_first;  /* Search first in UMB */
    int Cur_Strategy;   /* Current search strategy */

    /*-- Screen layout -----*/

    clrscr();
    Print( 1, 1, 0x07,
            "DOS Memory Management Demo " );
    Print( 51, 1, 0x07,
            " (C) 1991 by Michael Tischer" );
    Print( 1, 2, 0x07, "===== " );
    Print( 40, 2, 0x07, "===== " );
    Print( 25, 5, 0x07, "Processing memory layout now..." );

    /*-- Store DOS values when program starts -----*/

    StartStrategy = ReadStrategy(); /* Memory layout strategy */
    StartUMB = ReadUMB(); /* UMB inclusion */
    Allocate_Memory(); /* Create test environment */
    SetDosStrategy( StartStrategy ); /* Restore old strategy */
    SetUMB( StartUMB );

    if ( ConvSeg == 0 ) /* Allocate conventional memory? */
    { /* No --> End program with error message */
        clrscr();

```

```

    printf( "MEMDEMO: Not enough memory!\n" );
    exit(1);
}

/*-- Starting values for memory management -----*/
Cur_UMB_INC = ( StartUMB == UMB_INC );
Cur_UMB_first = ( ( StartStrategy & SFIRST_UMB ) == SFIRST_UMB );
Cur_Strategy = ( StartStrategy & ( 0xFF ^ SFIRST_UMB ) );

/*-- Demonstration of memory division -----*/

clrscr();
Print( 1, 1, 0x07,
      "DOS Memory Management Demo " );
Print( 51, 1, 0x07,
      " (C) 1991 by Michael Tischer" );
Print( 1, 2, 0x07, "===== " );
Print( 40, 2, 0x07, "===== " );
Demo( Cur_UMB_INC, Cur_UMB_first, Cur_Strategy );

/*-- Restore old DOS values -----*/

ReleaseThisMemory();          /* Release reserved memory */
SetDosStrategy( StartStrategy );
SetUMB( StartUMB );
clrscr();
}

```