

```

***** M E M D E M O P . P A S *****
*-----*
* Task : Demonstrates DOS memory management. *
*-----*
* Author : Michael Tischer *
* Developed on : 10/08/91 *
* Last update : 01/29/92 *
*****
{$M 8096, 0, 10240 }

program MEMDEMOP;

uses crt, { Add CRT and DOS units }
dos;

{== Constants =====}

const {-- Function numbers for interrupt 21H -----}

GET_MEM = $48; { Reserve RAM }
FREE_MEM = $49; { Release RAM }
CHANGE_MEM = $4A; { Change size of a memory range }
GET_STRATEGY = $5800; { Get memory division strategy }
SET_STRATEGY = $5801; { Set memory division strategy }
GET_UMB = $5802; { Get UMB }
SET_UMB = $5803; { Set UMB }

{-- Search strategies for SET_STRATEGY -----}

SRCHS_BELOW = $00; { Use the first free memory block }
SRCHS_BEST = $01; { Use the best memory block }
SRCHS_ABOVE = $02; { Use the last free memory block }
SFIRST_UMB = $80; { Search for first in UMB }
{ (combined with SEARCH_...) }

{-- Constants for SetUMB -----}

UMB_OUT = $00; { Ignore UMB }
UMB_INC = $01; { Include UMB in memory allocation }

{-- Constants for Demo -----}

TEST_EN = 10240-1; { 10239 paragraph test environment }
TEST_EN_UMB = 2560-1; { 2559 paragraph UMB test environment }
TEST_EN_KB = 160; { Test environment is 160K }
TEST_EN_UMB_KB = 40; { Test environment UMB is 40K }
BLOCKAMT = 26; { Number of addresses for result display }

{-- Key codes for control -----}

ESC = #27; { Press <Esc> to cancel }
F1 = #59; { Function key F1 }
F2 = #60; { Function key F2 }
F3 = #61; { Function key F3 }
F8 = #66; { Function key F8 }
F9 = #67; { Function key F9 }
F10 = #68; { Function key F10 }

{== Type declarations =====}

type BlockType = record { Manage an allocated memory block }
SegAddr, { Segment address }
MBLSize : word; { Memory block size in K }
end;

{== Typed constants =====}

const YesNo : array[ false..true ] of string =
( 'No ', 'Yes' );
SText : array[ SRCHS_BELOW..SRCHS_ABOVE ] of string =
( 'First free memory block used',
'Best free memory block used ',
'Last free memory block used ' );
ColArray : array[ 0..1 ] of byte = ( $07, $70 );
Keybd_Text : array[ 0..3 ] of string =
( ' [F1] Allocate memory',
' [F2] Release memory',
' [F3] Change size',
' [ESC] End program' );

{== Global variables =====}

var Regs : Registers; { Registers for interrupt call }
MAddrArray : array[ 0..1000 ] of word; { Array for memory }
Num_Addrs : word; { Number of addresses }

```

```

ConvSeg      : word;      { Address of test block in conventional RAM }
UMBSeG      : word;      { Address of test block in UMB }
BlocArray   : array[ 0..BLOCKAMT - 1 ] of BlockType; { Memory }

```

```

{*****}
{ 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 }
{*****}

```

```

procedure DOS_GetMem(      Ps : word;
                        var Adr : word;
                        var Res : word );

```

```

begin
  with Regs do
    begin
      ah := GET_MEM; { Function number }
      bx := Ps;      { Number of paragraphs to be reserved }
      msdos( Regs ); { Interrupt call }
      if ( flags and fcarry = 0 ) then { Call successful? }
        begin
          Adr := regs.ax; { Yes --> Return address and size }
          Res := Ps;
        end
      else { No --> Error }
        begin
          Adr := 0; { No memory reserved }
          Res := bx; { Maximum available size }
        end;
      end;
    end;
end;

```

```

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

```

```

procedure DOS_FreeMem( Adr : word );

```

```

begin
  with Regs do
    begin
      ah := FREE_MEM; { Function number }
      es := Adr;      { Address of memory to be released }
      msdos( Regs ); { Interrupt call }
    end;
  end;
end;

```

```

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

```

```

procedure DOS_ChangeMem(      Ps : word;
                        var Adr : word;
                        var Res : word );

```

```

begin
  with Regs do
    begin
      ah := CHANGE_MEM; { Function number }
      bx := Ps;          { Number of paragraphs to be reserved }
      es := Adr;         { Segment address of memory block to be changed }
      msdos( Regs );     { Interrupt call }
      if ( flags and fcarry = 0 ) then { Call successful? }
        Res := Ps { Yes --> New memory size }
      else { No --> Error }
        Res := bx; { Maximum available memory size }
      end;
    end;
  end;
end;

```

```

{*****}
{ Read_Strategy : Reads the DOS memory management strategy. }
{ Input         : None }
{ Output        : Memory management strategy }
{*****}

```

```

function Read_Strategy : integer;

```

```

begin
  with Regs do
    begin
      ax := GET_STRATEGY;           { Set function number }
      msdos( Regs );                 { Interrupt call }
      Read_Strategy := ax;           { Display strategy }
    end;
  end;

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

function ReadUMB : integer;

begin
  with Regs do
    begin
      ax := GET_UMB;                 { Set function number }
      msdos( Regs );                 { Interrupt call }
      ReadUMB := al;                 { Display status }
    end;
  end;

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

procedure SetDosStrategy( Strategy : integer );

begin
  with Regs do
    begin
      ax := SET_STRATEGY;           { Set function number }
      bx := Strategy;               { Interrupt call }
      msdos( Regs );
    end;
  end;

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

procedure SetUMB( UMB : integer );

begin
  with Regs do
    begin
      ax := SET_UMB;                 { Set function number }
      bx := UMB;                     { Interrupt call }
      msdos( Regs );
    end;
  end;

  {*****}
  { Allocate_Memory: Creates a test environment for the memory test. }
  { Input          : None }
  { Output         : None }
  {*****}

procedure Allocate_Memory;

var SegAdr      : word;              { Segment address of allocated memory }
    Des_Mem     : word;              { Desired memory size }
    MBlSize     : word;              { Size of allocated memory range }

begin
  {-- 1. Allocate test block -----}

  SetUMB( UMB_OUT );                 { Test block in conventional memory }
  DOS_GetMem( TEST_EN, ConvSeg, MBlSize ); { Get test block }
  if ( ConvSeg = 0 ) then              { Error? }
    exit;                             { No test block found, end procedure }

  SetUMB( UMB_INC );                 { Test block in UMB }
  SetDosStrategy( SRCHS_ABOVE or SFIRST_UMB );
  DOS_GetMem( TEST_EN_UMB, UMBSeg, MBlSize );
  if ( UMBSeg <> 0 ) and ( UMBSeg < $A000 ) then { No UMB available? }

```

```

begin
    DOS_FreeMem( UMBSeg );           { Release memory }
    UMBSeg := 0;                     { No UMBS }
end;

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

Des_Mem := 63;                      { First try allocating 15 paragraphs }
Num_Addrs := 0;                     { Set number of allocated 1K blocks to 0 }
repeat
    DOS_GetMem( Des_Mem, SegAdr, MBlSize ); { Call for memory }
    if ( SegAdr <> 0 ) then { Memory received? }
    begin
        MAddrArray[ Num_Addrs ] := SegAdr; { Note address }
        inc( Num_Addrs );
    end;
until ( SegAdr = 0 ); { Allocate all memory }

{-- 3. Release test block -----}

if ( ConvSeg > 0 ) then { Could memory be allocated? }
begin { Yes }
    DOS_FreeMem( ConvSeg );
    dec( ConvSeg ); { MCB is also free }
end;
if ( UMBSeg > 0 ) then { Could UMB be allocated? }
begin { Yes }
    DOS_FreeMem( UMBSeg );
    dec( UMBSeg ); { MCB is also free }
end;
end;

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

procedure ReleaseThisMemory;

var i : word; { Loop counter }

begin
    if ( Num_Addrs > 0 ) then { Was memory allocated? }
    for i := 0 to Num_Addrs - 1 do { Yes --> Release block by block }
        DOS_FreeMem( MAddrArray[ i ] );
    end;

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

procedure DisplayMemLayout( W_Borders : boolean );

var SArray : array[ 0..TEST_EN_KB - 1 ] of char;
    SArray_UMB : array[ 0..TEST_EN_UMB_KB - 1 ] of char;
    i,j : word; { Loop counter }
    Position : word; { Help variable }
    Sdummy : string;
    LastChar_M : char; { Memory for last character displayed }
    CurColor : byte; { Current output color }

begin
    fillchar( SArray[ 0 ], TEST_EN_KB, #32 ); { Initialize display array }
    fillchar( SArray_UMB[ 0 ], TEST_EN_UMB_KB, #32 );

    {-- Fill memory table -----}

    for i := 0 to BLOCKAMT - 1 do
        begin
            if ( BlocArray[ i ].SegAdr > $A000 ) then { UMB? }
            begin
                Position := ( BlocArray[ i ].SegAdr - UMBSeg ) div 64;
                for j := 0 to BlocArray[ i ].MBlSize div 64 do
                    SArray_UMB[ Position + j ] := chr( i + 65 );
                end
            else if ( BlocArray[ i ].SegAdr > 0 ) then
            begin
                Position := ( BlocArray[ i ].SegAdr - ConvSeg ) div 64;
                for j := 0 to BlocArray[ i ].MBlSize div 64 do
                    SArray[ Position + j ] := chr( i + 65 );
                end;
            end;
        end;
    end;
end;

```

[illegible]

```

Des_Mem : word;           { Size of reservation }
MBLSize : word;
sdummy  : string;

begin
  {-- Initialize address array -----}

  for i := 0 to BLOCKAMT - 1 do      { No more allocated blocks }
    with BlocArray[ i ] do
      begin
        SegAddr := 0;                { Segment address of memory block }
        MBLSize := 0;                { Size of memory block }
      end;
    end;

  gotoxy( 1, 7 );
  DisplayMemLayout( TRUE );          { Display memory table }

  {-- Demonstration loop -----}

  repeat
    {-- Specify selected memory management strategy -----}

    if ( Inc_UMB ) then                { UMB used? }
      SetUMB( UMB_INC )
    else
      SetUMB( UMB_OUT );

    if ( UMB_first ) then
      SetDosStrategy( Strategy or SFIRST_UMB )
    else
      SetDosStrategy( Strategy );

    {-- Display current concept -----}

    gotoxy( 1, 3 );
    writeln( 'Memory management strategy: ', SText[ Strategy ],
              ' [F8] ' );
    writeln( 'First search in UMB          : ', YesNo[ UMB_first ],
              ' [F9] ' );
    writeln( 'UMB used                    : ', YesNo[ Inc_UMB ],
              ' [F10] ' );
    writeln( '-----',
              '-----' );

    {-- Entry and processing -----}

    repeat until keypressed;           { Wait for a key }
    IKeys := readkey;                  { Read key }
    if ( ( IKeys = #0 ) and ( keypressed ) ) then { Function key }
      IKeys := readkey;                { Get 2nd key code }

    case IKeys of
      F1 :                             { Allocate memory block }
        begin
          i := -1;                      { No more valid memory blocks given }
          repeat
            gotoxy( 1, 23 );
            write( 'Reserve which block [ A-Z ]: ' );
            readln( ResPtr );
            ResPtr := upcase( ResPtr );
            if ( ResPtr >= 'A' ) and ( ResPtr <= 'Z' ) then
              if ( BlocArray[ ord( ResPtr ) - 65 ].SegAddr = 0 ) then
                i := ord( ResPtr ) - 65;
            until ( i <> -1 );
            write( 'How many K do you want reserved: ' );
            readln( 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 ) then { Error? }
              begin
                str( ( BlocArray[ i ].MBLSize + 1 ) div 64, sdummy );
                write( 'Only ' + sdummy + 'K free' );
                repeat until keypressed;
                while keypressed do
                  IKeys := readkey;
                  IKeys := #0;
                end;
                gotoxy( 1, 23 );
                writeln( ' ' );
                writeln( ' ' );
                write( ' ' );
                gotoxy( 1, 7 );
                DisplayMemLayout( FALSE ); { Display memory table }
              end;
          end;
        end;
    end;
  end;

```

[illegible]

```

{-- Store DOS values when program starts -----}

StartStrategy := Read_Strategy;           { Memory layout strategy }
StartUMB := ReadUMB;                      { UMB inclusion }
Allocate_Memory;                          { Create test environment }
SetDosStrategy( StartStrategy );          { Restore old strategy }
SetUMB( StartUMB );

if ( ConvSeg = 0 ) then                   { Allocate conventional memory? }
begin                                   { No --> End program with error message }
    clrscr;
    writeln('MEMDEMOP: Not enough memory!');
    exit;                               { End program }
end;

{-- Starting values for memory management -----}

Cur_UMB_inc := ( StartUMB = UMB_INC );
Cur_UMB_first := ( ( StartStrategy and SFIRST_UMB ) = SFIRST_UMB );
Cur_Strategy := StartStrategy and ( $FF xor SFIRST_UMB );

{-- Demonstration of memory division -----}

clrscr;
writeln( 'DOS Memory Management Demo ',
        ' (C) 1991 by Michael Tischer' );
writeln( '===== ',
        '===== ' );
Demo( Cur_UMB_inc, Cur_UMB_first, Cur_Strategy );

{-- Restore old DOS values -----}

ReleaseThisMemory;                       { Release reserved memory }
SetDosStrategy( StartStrategy );
SetUMB( StartUMB );
clrscr;
end.

```