

```

***** M C B P . P A S *****
*-----*
* Task : Displays any memory block allocated by DOS. *
*-----*
* Author : Michael Tischer *
* Developed on : 08/22/88 *
* Last update : 01/29/92 *
*****

program MCBP;

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

type BytePtr = ^byte; { Pointer to a byte }
MemRnge = array[0..1000] of byte; { A range somewhere in RAM }
RngPtr = ^MemRnge; { Pointer to a range }
MCB = record { A Memory Control Block }
    IdCode : char; { "M" = block follows, "Z" = end }
    PSP : word; { Segment address of appropriate PSP }
    Spacing : word; { Number of paragraphs - 1 }
end;
MCBPtr = ^MCB; { Pointer to an MCB }
MCBPtr2 = ^MCBPtr; { Pointer to an MCBPtr }
HexStr = string[4]; { Stores a 4-digit hex string }

var CvHStr : HexStr; { Stores the hex string after conversion }

{*****}
{ * HexString: Converts a number into a hexadecimal string. * }
{ * Input : - HexVal = Value to be converted * }
{ * Output : The converted hex string * }
{*****}

function HexString( HexVal : word ) : HexStr;

var counter, { Loop counter }
Nibble : byte; { Lower nibble of word }

begin
    CvHStr := 'xxxx'; { Generate a string }
    for counter:=4 downto 1 do { Get the 4 numbers in the string }
        begin
            Nibble := HexVal and $000f; { Get the upper 4 bits only }
            if ( Nibble > 9 ) then { Convert to a character? }
                CvHStr[ counter ] := chr(Nibble - 10 + ord('A')) { Yes }
            else { No --> Convert to number }
                CvHStr[ counter ] := chr(Nibble + ord('0'));
            HexVal := HexVal shr 4; { HexVal 4 bit positions to the right }
        end;
    HexString := CvHStr; { Pass the resulting string }
end;

{*****}
{ * FirstMCB: Returns a pointer to the first MCB. * }
{ * Input : None * }
{ * Output : Pointer to the first MCB * }
{*****}

function FirstMCB : MCBPtr;

var Regs : Registers; { Store the processor registers }

begin
    Regs.ah := $52; { Func. no.: Get DOS info block's address }
    MsDos( Regs ); { Call DOS interrupt 21H }

    { *-- ES:(BX-4) points to the first MCB, create pointer -----* }

    FirstMCB := MCBPtr2( ptr( Regs.ES-1, Regs.BX+12 ) )^;
end;

{*****}
{ * Dump : Displays a memory range as hex and ASCII dumps. * }
{ * Input : - DPTr = Pointer to the memory range to be dumped * }
{ * : - NumL = Number of 16-byte lines to be dumped * }
{ * Output : None * }
{*****}

procedure Dump( DPTr : RngPtr; NumL : byte);

type HBStr = string[2]; { Get 2-digit hex number }

var Offset, { Offset in memory range }
Z : integer; { Loop counter }
HexStr : HBStr; { Create a hex string for hex dump }

```

```

procedure HexByte( HByte : byte );
begin
  HexStr[1] := chr( (HByte shr 4) + ord('0') ); { First digit }
  if HexStr[1] > '9' then { Convert to character? }
    HexStr[1] := chr( ord(HexStr[1]) + 7 ); { Yes }
  HexStr[2] := chr( (HByte and 15) + ord('0') ); { Second digit }
  if HexStr[2] > '9' then { Convert to character? }
    HexStr[2] := chr( ord(HexStr[2]) + 7 ); { Yes }
end;

begin
  HexStr := 'zz'; { Generate hex string }
  writeln;
  write('DUMP 3 0123456789ABCDEF 00 01 02 03 04 05 06 07 08');
  writeln(' 09 0A 0B 0C 0D 0E 0F');
  write('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA');
  writeln('AAAAAAAAAAAAAAAAAAAAAAAAAAAA');
  Offset := 0; { Start with first byte in the range }
  while NumL > 0 do { Execute loop NumL times }
  begin
    write(HexString(Offset), ' 3 ');
    for Z:=0 to 15 do { Process 15 bytes }
      if (Dptr^[Offset+Z] >= 32) then { Valid ASCII characters? }
        write( chr(Dptr^[Offset+Z]) ) { Yes --> Display }
      else { No }
        write(' '); { Display a space instead of a character }
    write(' '); { Place cursor at hex section }
    for Z:=0 to 15 do { Process 15 bytes }
    begin
      HexByte( Dptr^[Offset+Z] ); { Convert byte to hex }
      write(HexStr, ' '); { Display hex string }
    end;
    writeln;
    Offset := Offset + 16; { Set offset at the next line }
    Dec( NumL ); { Decrement the number of remaining lines }
  end;
  writeln;
end;

```

```

{*****}
{* TraceMCB: Display list of MCBs. *}
{* Input : None *}
{* Output : None *}
{*****}

```

```

procedure TraceMCB;

const ComSpec : array[0..7] of char = 'COMSPEC=';

var CurMCB : MCBPtr;
    EndIt : boolean;
    PdKey : char;
    NrMCB, { Number of MCBs to be checked }
    Z, { Loop counter }
    MemPtr : RngPtr;

begin
  EndIt := false;
  NrMCB := 1; { Assign the first MCB the number 1 }
  CurMCB := FirstMCB; { Get pointer to the first MCB }
  repeat { Add to group of MCBs }
  if CurMCB^.IdCode = 'Z' then { Last MCB reached? }
    EndIt := true; { Yes }
  writeln('MCB number = ', NrMCB);
  writeln('MCB address = ', HexString(seg(CurMCB^)), ':',
    HexString(ofs(CurMCB^)) );
  writeln('Memory address = ', HexString(succ(seg(CurMCB^))), ':',
    HexString(ofs(CurMCB^)) );
  writeln('ID = ', CurMCB^.IdCode);
  writeln('PSP address = ', HexString(CurMCB^.PSP), ':0000');
  writeln('Size = ', CurMCB^.Spacing, ' paragraphs ',
    '( ', longint(CurMCB^.Spacing) shl 4, ' bytes )');
  write('Contents = ');

  {*- Handle MCB as an environment? -----*}

  Z := 0; { Start comparison with first byte }
  MemPtr := RngPtr(ptr(seg(CurMCB^)+1, 0)); { Pointer in RAM }
  while ( (Z<=7) and (ord(ComSpec[Z]) = MemPtr^[Z]) ) do
    Inc(Z); { Set Z at the next character }
  if Z>7 then { String found? }
  begin { Yes --> Handle as an environment }
    writeln('Environment');
    MemPtr := RngPtr(ptr(seg(CurMCB^)+1, 0));
  end;
end;

```

