

# Listing: MCBP.PAS

```
{*****}
{*                                     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 : 04/07/95 *}
{*****}
```

```
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 *}
{*****}
```



```

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    : Display 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 | 0123456789ABCDEF          00 01 02 03 04 05 06 07 08');
writeln(' 09 0A 0B 0C 0D 0E 0F');
write('-----+-----');
writeln('-----');
Offset := 0;                                     { Start with first byte in the range }
while NumL>0 do                                  { Execute loop NumL times }
begin
write(HexString(Offset), ' | ');
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,      : integer;           { Number of MCBs to be checked }
     Z           : integer;           { 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));
    if Lo(DosVersion) >= 3 then          { DOS Version 3.0 or higher? }
      begin                             { Yes --> List program name }
        write('Program name   = ');
        Z := 0;                         { Start with the first byte }
        while not( (MemPtr^[Z]=0) and (MemPtr^[Z+1]=0) ) do
          Inc( Z );                     { Search for null string }
        if ( MemPtr^[Z+2]=1 ) and ( MemPtr^[Z+3]=0 ) then
          begin                         { Program name found }
            Z := Z + 4; { Place Z at first character of prg. name }
            repeat                               { Show program names }
              write( chr(MemPtr^[Z]) );          { Display character }
              Inc( Z );                          { Process next character }
            until ( MemPtr^[Z]=0 );              { until end of string }
            writeln;
          end
        else                                     { No program name found }
          writeln('Unknown');
        end;
      end;
    end;
  end;
end;

```

```

{*- Display environment string -----*}

writeln(#13,#10, 'Environment string');
Z := 0;           { Start with first byte in allocated range }
while MemPtr^[Z]<>0 do           { Repeat until null string }
begin
  write('      ');
  repeat
    write( chr(MemPtr^[Z]) );           { Display string }
    Inc( Z );                           { Display a character }
    until MemPtr^[Z]=0;                 { Process next character }
    Inc( Z );                           { until end of string }
    writeln;                           { Set to start of next string }
  end
end
else
begin
  { No environment }

  {*- Handle it as a PSP? -----*}
  {*- (If INT 20 (Code=$CD $20) starts the code) -----*}

  MemPtr := RngPtr(ptr(seg(CurMCB^)+1, 0));           { Set pointer }
  if ( (MemPtr^[0]=$CD) and (MemPtr^[1]=$20) ) then
  begin
    writeln('PSP (with program following)');           { Handled as a PSP }
  end
  else
  begin
    { INT 20H could not be implemented }
    writeln('Unidentifiable as program or data');
    Dump( MemPtr, 5);           { Dump the first 5x16 bytes }
  end
end;
end;

```

```

write('-----');
writeln('----- Please press a key -----');
if ( not EndIt ) then
  begin
    CurMCB := MCBPtr(ptr(seg(CurMCB^) + CurMCB^.Spacing + 1, 0)); { Set pointer to the next MCB }
    Inc(NrMCB); { Increment the number of MCBs }
    PdKey := ReadKey;
  end;
until ( EndIt ) { Repeat until the last MCB is processed }
end;

{ ***** }
{ **                MAIN PROGRAM                ** }
{ ***** }

begin
  ClrScr; { Clear screen }
  writeln( 'MCBP - (c) 1988, 92 by Michael Tischer' );
  writeln;
  writeln;
  TraceMCB; { Display MCB group }
end.

```