

Pascal listing: DUMPP.PAS

```
{*****}
{*          D U M P P          *}
{*-----*}
{* Task          : A filter, which reads in characters from the *}
{*               standard input device and outputs them as a   *}
{*               hex and ASCII dump on the standard output device *}
{*-----*}
{* Author       : MICHAEL TISCHER                               *}
{* Developed on  : 08/08/87                                       *}
{* Last update   : 04/07/95                                       *}
{*-----*}
{* Info          : This program can only be called from the    *}
{*               DOS level after compiling to an EXE file       *}
{*               with TURBO                                       *}
{*-----*}
{*****}
```

```
program DUMPP;
```

```
Uses Dos;                                { Add DOS unit }
```

```
{ $V- }                                { Suppress length test on strings }
```

```
const NUL = 0;      { ASCII code for NULL character }
      BEL = 7;      { ASCII code for BELL           }
      BS  = 8;      { ASCII code for Backspace      }
      TAB = 9;      { ASCII code for Tab            }
      LF  = 10;     { ASCII code for Linefeed       }
      CR  = 13;     { ASCII code for Carriage Return }
      EOF = 26;     { ASCII code for End of File    }
      ESC = 27;     { ASCII code for Escape         }
```



```
procedure DoDump;
```

```
var Regs      : Registers;           { Register variable for interrupt call }
    NineBytes: array[1..9] of char; { Accepts the characters read in }
    DumpBuf   : DumpBf;              { Accepts a line for dumping }
    HexChr,
    Counter,
    NextA     : integer;              { Pointer in buffer for ASCII code }
    Endc      : boolean;              { Another byte read in? }
```

```
begin
```

```
    Endc := false;                    { Not the end }
```

```
    repeat
```

```
        Regs.ah := $3F;                { Function number for reading handle }
```

```
        Regs.bx := 0;                  { Standard input device is handle 0 }
```

```
        Regs.cx := 9;                  { Read in 9 characters }
```

```
        Regs.ds := seg(NineBytes);     { Segment address of the buffer }
```

```
        Regs.dx := ofs(NineBytes);     { Offset address of the buffer }
```

```
        MsDos( Regs );                 { Call DOS interrupt 21H }
```

```
        if (Regs.ax = 0) then Endc := true; { No character read? }
```

```
        if not(Endc) then
```

```
            begin                                { No }
```

```
                for Counter := 1 to 30          { Fill buffer with spaces }
```

```
                    do DumpBuf [Counter] := ' ';
```

```
                DumpBuf[31] := #219;            { Place separator between hex and ASCII }
```

```
                NextA := 32;                    { ASCII characters follow separator }
```

```
                for Counter := 1 to Regs.ax do  { Start processing characters }
```

```
                    begin                                { Read in }
```

```
                        HexChr := ord(NineBytes[Counter]) shr 4 + 48; { Hex top 4 bits }
```

```
                        if (HexChr > 57) then HexChr := HexChr + 7;      { Convert char }
```

```
                        DumpBuf[Counter * 3 - 2] := chr(HexChr);       { Store in buffer }
```

```
                        HexChr := ord(NineBytes[Counter]) and 15 + 48;
```

```

if (HexChr > 57) then HexChr := HexChr + 7;      { Hex bottom 4 bits }
DumpBuf[Counter * 3 - 1] := chr(HexChr);        { Convert number }
case ord(NineBytes[Counter]) of                 { Store in buffer }
  NUL : SZ(DumpBuf, 'NUL', NextA);              { Test ASCII code }
  BEL : SZ(DumpBuf, 'BEL', NextA);              { NULL }
  BS  : SZ(DumpBuf, 'BS' , NextA);              { BELL }
  TAB : SZ(DumpBuf, 'TAB', NextA);              { Backspace }
  LF  : SZ(DumpBuf, 'LF' , NextA);              { Tab }
  CR  : SZ(DumpBuf, 'CR' , NextA);              { Linefeed }
  EOF : SZ(DumpBuf, 'EOF', NextA);              { Carriage Return }
  ESC : SZ(DumpBuf, 'ESC', NextA);              { End of File }
else                                             { Escape }
begin
  DumpBuf[NextA] := NineBytes[Counter];         { Normal character }
  NextA := succ(NextA)                          { Store ASCII char. }
end;                                           { Set pointer to next character }
end;
end;
DumpBuf[NextA] := #219;                        { Set end character }
DumpBuf[NextA+1] := chr(CR);                   { Carriage return followed }
DumpBuf[NextA+2] := chr(LF);                   { by linefeed to buffer end }
Regs.ah := $40;                                { Function number for writing handle }
Regs.bx := 1;                                  { Standard output device is handle 1 }
Regs.cx := NextA+2;                            { Number of characters }
Regs.ds := seg(DumpBuf);                       { Segment address of the buffer }
Regs.dx := ofs(DumpBuf);                       { Offset address of the buffer }
MsDos( Regs );                                 { Call DOS interrupt 21H }
end;
until Endc;                                   { Repeat until no more characters are available }
end;

```

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

```
begin
    DoDump;           { Output dump }
end.
```