

```

***** T S R P *****
*-----*
* Task      : Creates a TSR program with the help of an *
*            assembly language module.                  *
*-----*
* Author     : Michael Tischer                          *
* Developed on : 08/18/88                               *
* Last update  : 02/04/92                              *
*****

program TSRP;

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

{$M 2048, 0, 5120}      { Reserve 2K for the stack and 5K for the heap }
{$L tsrpa}              { Link assembler module }

{-- Declare external functions in the assembler module -----}

procedure TsrInit( PrcPtr   : word;   { Offset address: TSR procedure }
                  ResPara   : word;   { Number of pars. to be reserved }
                  ) ; external ;      { ID string }

function TsrIsInst( i2F_fctno : byte ) : boolean ; external ;
procedure TsrUnInst; external;       { Uninstall TSR }
procedure TsrSetPtr( Offset : word ); external;
function TsrCanUnInst : boolean; external;

{$F+}                                { FAR procedures and functions }
procedure TsrCall ; external;
procedure TsrSetHotKey( KeyMask : word;      { Hotkey (see CONST) }
                      ScCode   : byte;      { Scan code }
                      ) ; external;

{$F-}

{-- Constants -----}

{-- Scan codes for different keys -----}

const SC_ESC          = $01;      SC_Z          = $2C;
      SC_1            = $02;      SC_X          = $2D;
      SC_2            = $03;      SC_C          = $2E;
      SC_3            = $04;      SC_V          = $2F;
      SC_4            = $05;      SC_B          = $30;
      SC_5            = $06;      SC_N          = $31;
      SC_6            = $07;      SC_M          = $32;
      SC_7            = $08;      SC_COMMA      = $33;
      SC_8            = $09;      SC_PERIOD     = $34;
      SC_9            = $0A;      SC_SLASH      = $35;
      SC_0            = $0B;      SC_SHIFT_RIGHT = $36;
      SC_HYPHEN       = $0C;      SC_asterisk    = $37;
      SC_EQUALS       = $0D;      SC_ALT        = $38;
      SC_BACKSPACE    = $0E;      SC_SPACE      = $39;
      SC_TAB          = $0F;      SC_CAPS       = $3A;
      SC_Q            = $10;      SC_F1         = $3B;
      SC_W            = $11;      SC_F2         = $3C;
      SC_E            = $12;      SC_F3         = $3D;
      SC_R            = $13;      SC_F4         = $3E;
      SC_T            = $14;      SC_F5         = $3F;
      SC_Y            = $15;      SC_F6         = $40;
      SC_U            = $16;      SC_F7         = $41;
      SC_I            = $17;      SC_F8         = $42;
      SC_O            = $18;      SC_F9         = $43;
      SC_P            = $19;      SC_F10        = $44;
      SC_LBRACKET     = $1A;      SC_NUM_LOCK   = $45;
      SC_RBRACKET     = $1B;      SC_SCROLL_LOCK = $46;
      SC_ENTER        = $1C;      SC_CURSOR_HOME = $47;
      SC_CONTROL       = $1D;      SC_CURSOR_UP   = $48;
      SC_A            = $1E;      SC_CURSOR_PG_UP = $49;
      SC_S            = $1F;      SC_NUM_MINUS   = $4A;
      SC_D            = $20;      SC_CURSOR_LEFT  = $4B;
      SC_F            = $21;      SC_NUM_5       = $4C;
      SC_G            = $22;      SC_CURSOR_RIGHT = $4D;
      SC_H            = $23;      SC_NUM_PLUS    = $4E;
      SC_J            = $24;      SC_CURSOR_END   = $4F;
      SC_K            = $25;      SC_CURSOR_DOWN  = $50;
      SC_L            = $26;      SC_CURSOR_PG_DOWN = $51;
      SC_SEMICOLON     = $27;      SC_INSERT    = $52;
      SC_APOSTROPHE    = $28;      SC_DELETE    = $53;
      SC_GRAVE         = $29;      SC_SYS_REQUEST = $54;
      SC_SHIFT_LEFT    = $2A;      SC_F11       = $57;
      SC_BACKSLASH     = $2B;      SC_F12       = $58;
      SC_NOKEY         = $80;      { No more keys }

{-- Bit masks for the different toggle keys -----}

```

```

LSHIFT = 1; { Left SHIFT key }
RSHIFT = 2; { Right SHIFT key }
CTRL = 4; { CTRL key }
ALT = 8; { ALT key }
SYSREQ = 1024; { SYS-REQ key (AT keyboard only) }
BREAK = 4096; { BREAK key }
NUM = 8192; { NUM LOCK key }
CAPS = 16384; { CAPS LOCK key }
INSERT = 32768; { INSERT key }

I2F_CODE = $C4; { Function number INT 2F }
I2F_FCT_0 = $AA; { Code for INT 2F, function 0 }
I2F_FCT_1 = $BB; { Code for INT 2F, function 1 }

{-- Type declarations -----}

type VBuf = array[1..25, 1..80] of word; { Describes the screen }
VPtr = ^VBuf; { Pointer to a screen buffer }

{-- Declaration of procedure and function types which copy --}
{-- procedures and functions in the already installed copy --}
{-- of the TSR. --}

WoAPrcK = procedure; { Procedure without arguments }
SHKPrck = procedure( KeyMask : word; { TsrSetHotkey }
                    ScCode : byte );
PPtrT = record { Union for creating the procedure pointer }
        case integer of
            1 : ( WoAPrc : WoAPrcK );
            2 : ( SHKPrck : SHKPrck );
        end;

const Call : PPtrT = ( WoAPrc : TsrCall );

{-- Global variables -----}

var MBuf : VBuf absolute $B000:0000; { Monochrome video RAM }
    CBuf : VBuf absolute $B800:0000; { Color video RAM }
    VioPtr : VPtr; { Pointer to the video RAM }
    ATimes : integer; { Number of TSR activations }

{*****}
{ * DispInit: Creates a pointer to video RAM. * }
{ * Input : None * }
{ * Output : None * }
{*****}

procedure DispInit;

var Regs: Registers; { Store the processor registers }

begin
    Regs.ah := $0f; { Funct. no. 15 = Read the video mode }
    Intr($10, Regs); { Call the BIOS video interrupt }
    if Regs.al=7 then { Monochrome video card? }
        VioPtr := @MBuf { Yes --> Set pointer to monochrome video RAM }
    else { Handle it as EGA, VGA or CGA }
        VioPtr := @CBuf; { Set pointer to color video RAM }
    end;

{*****}
{ * SaveScreen: Saves the screen contents to a buffer. * }
{ * Input : SPTR : Pointer to the buffer in which the screen * }
{ * contents will be saved. * }
{ * Output : None * }
{*****}

procedure SaveScreen( SPtr : VPtr );

var scrow, { Current row }
    column : byte; { Current column }

begin
    for scrow:=1 to 25 do { Execute the 25 screen rows }
        for column:=1 to 80 do { Execute the 80 screen columns }
            SPtr^[scrow, column] := VioPtr^[scrow, column]; { Store char. }
        { & attribute }
    end;

{*****}
{ * RestoreScreen: Copies the contents of a buffer to video RAM. * }
{ * Input : BPTR : Pointer to the buffer whose contents are to be * }
{ * copied to video RAM * }
{ * Output : None * }
{*****}

```

```

procedure RestoreScreen( BPtr: VPtr );
var scrow,
    column : byte;
begin
    for scrow:=1 to 25 do
        for column:=1 to 80 do
            VioPtr^[scrow, column] := BPtr^[scrow, column];
        end;
    end;

{*****}
{ * ResPara: Calculates the number of paragraphs which must be          * }
{ * allocated for the program.                                          * }
{ * Input      : None                                                  * }
{ * Output    : The number of paragraphs to be reserved                * }
{*****}

function ResPara : word;
begin
    {-- Compute the number of bytes needed, using the proper method ----}

    {$ifdef VER50}
        ResPara := Seg(FreePtr^)+$1000-PrefixSeg; { Number of paragraphs }
    {$endif}

    {$ifdef VER55}
        ResPara := Seg(FreePtr^)+$1000-PrefixSeg; { Number of paragraphs }
    {$endif}

    {$ifdef VER60}
        ResPara := Seg(HeapEnd^)-PrefixSeg;
    {$endif}
end;

{*****}
{ * ParamGetHotKey: Checks command line parameters for the hotkey      * }
{ * switch (/T) and implements these keys.                             * }
{ * Input      : KEYMASK = Variable for storing the key mask           * }
{ *             SCCODE  = Variable for storing the scan code           * }
{ * Output    : TRUE if the hotkeys are supported, otherwise FALSE     * }
{ * Info      : - Parameters not beginning with /T are ignored as      * }
{ *             parameters, but may be handled as other routines       * }
{ *             - If no parameter exists for /T, SC_NOKEY is placed in * }
{ *             the appropriate variable.                               * }
{*****}

function ParamGetHotKey( var KeyMask : word;
                        var ScCode : byte ) : boolean;

type ToggleKey = record
    Name : string[6];
    WVal : word;
end;

const TogKeys : array[ 1..9 ] of ToggleKey =
    ( ( Name: 'LSHIFT'; WVal : LSHIFT ),
      ( Name: 'RSHIFT'; WVal : RSHIFT ),
      ( Name: 'CTRL'; WVal : CTRL ),
      ( Name: 'ALT'; WVal : ALT ),
      ( Name: 'SYSREQ'; WVal : SYSREQ ),
      ( Name: 'BREAK'; WVal : BREAK ),
      ( Name: 'NUM'; WVal : NUM ),
      ( Name: 'CAPS'; WVal : CAPS ),
      ( Name: 'INSERT'; WVal : INSERT )
    );

var i , j,
    code,
    dummy : integer;
    arg : string;
begin
    KeyMask := 0;
    ScCode := SC_NOKEY;

    for i := 1 to ParamCount do
        arg := ParamStr(i);
        for j := 1 to length(arg) do
            arg[j] := upcase(arg[j]);
        end;
        if ( arg[1] = '/' ) and ( arg[2] = 'T' ) then
            delete( arg, 1, 2 );
        end;
    end;
end;

```

```

val( arg, code, dummy );      { Convert remainder to binary }
if ( dummy = 0 ) then        { Conversion O.K.? }
begin                        { Yes }
  if ( code > 0 ) and ( code < 128 ) then { Valid code? }
    ScCode := Code           { Yes --> Store }
  else
    begin                    { Invalid code }
      ParamGetHotKey := false;
      exit;                  { End function with FALSE }
    end;
end
else { If not a number, must be the name of a toggle key }
begin
  j := 1;                    { Search toggle key array }
  while ( j < 10 ) and ( arg <> TogKeys[j].Name ) do
    j := j + 1;
  if ( j < 10 ) then          { Name found? }
    KeyMask := KeyMask or TogKeys[j].WVal { Yes --> Flag }
  else
    begin                    { No --> Neither number nor toggle key }
      ParamGetHotKey := false;
      exit;                  { End function with FALSE }
    end;
end;
end;
end;
ParamGetHotKey := true;      { If everything checks out }
end;                          { so far, parameters are O.K. }

```

```

{*****}
{ * EndTPrc: Called by the assembler module when the TSR program is * }
{ *      uninstalled. * }
{ * Input   : None * }
{ * Output  : None * }
{ * Info    : This procedure must be a FAR procedure to permit access * }
{ *      from the installed copy of the TSR. * }
{*****}

```

```

{$F+}

procedure EndTPrc;

begin
  TextBackground( Black );      { Dark background }
  TextColor( LightGray );      { Light text }
  writeln('The TSR was called ', ATimes, ' times.');
```

```

end;

{$F-}

{*****}
{ * Tsr: Called by the assembler module after the hotkey is pressed. * }
{ * Input   : None * }
{ * Output  : None * }
{ * Info    : This procedure must be in the main program and may not * }
{ *      be turned into a FAR procedure by the $F+ compiler * }
{ *      directive. * }
{*****}

{$F-}                                { Don't make a FAR procedure }

```

```

procedure Tsr;

var BufPtr : VPtr;             { Stores pointer to the allocated blocks }
    Column,                               { The current screen column }
    ScRow : byte;                { The current screen row }
    PdKey : char;

begin
  while KeyPressed do          { Clear keyboard buffer }
    PdKey := ReadKey;
  inc( ATimes );               { Increment call counter }
  DispInit;                    { Get video RAM address }
  GetMem(BufPtr, SizeOf(VBuf) ); { Allocate buffer }
  SaveScreen( BufPtr );        { Store screen contents }
  ScRow := WhereY;              { Get current screen row }
  Column := WhereX;             { Get current screen column }
  TextBackground( LightGray ); { Light background }
  TextColor( Black );           { Dark text }
  ClrScr;                       { Clear screen }
  GotoXY(34, 12);
  write('My first TSR.');
```

```

FreeMem( BufPtr, SizeOf(VBuf) );           { Release allocated buffer }
GotoXY( Column, ScRow );                   { Return cursor to original position }
end;

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

var KeyMask : word;
    ScCode : byte;

begin
  writeln('TSRP - (c) 1988, 92 by Michael Tischer');
  if not ParamGetHotKey( KeyMask, ScCode ) then
    begin
      { Error in command line parameters }
      writeln( 'Illegal command line parameters' );
      exit;
    end;

  {-- Command line parameters were O.K. -----}

  if ( TsrIsInst( I2F_CODE ) = FALSE ) then
    { Program already
      { installed?
      { No }
    begin
      ATimes := 0;           { Program hasn't been enabled yet }
      writeln( 'TSR now installed. ' );
      if ( KeyMask = 0 ) and ( ScCode = SC_NOKEY ) then { No params.? }
        begin
          { No --> Default is ALT-H }
          TsrSetHotkey( ALT, SC_H );
          writeln( 'Press <ALT> + H to enable' );
        end
      else
        { Install user-defined hotkeys }
        TsrSetHotkey( KeyMask, ScCode );
        TsrInit( ofs(Tsr), ResPara );
        { Install program }
      end
    else
      { Program already installed }
    begin
      if ( KeyMask = 0 ) and ( ScCode = SC_NOKEY ) then { No params.? }
        begin
          { No --> Try uninstalling }
          if TsrCanUnInst then
            begin
              TsrSetPtr(ofs(EndTPrc));
              Call.WoAPrc;
              TsrUnInst;
              writeln( 'Program now uninstalled.' );
            end
          else
            writeln( 'Program cannot be uninstalled.' );
          end
        else
          { Implement new hotkey }
        begin
          writeln( 'New hotkey implemented' );
          TsrSetPtr(ofs(TsrSetHotKey));
          Call.SHKPrC( KeyMask, ScCode );
        end
      end;
    end;
  end.

```