

C listing: TSRC.C

```

/*****
/*
/*-----T S R C-----*/
/* Task : Creates a TSR program with the help of an assembly language module. */
/*-----*/
/* Author : Michael Tischer */
/* Developed on : 08/15/88 */
/* Last update : 04/06/95 */
/*-----*/
/* Memory model : SMALL */
/* Compilation (Microsoft C) */
/* CL /AS /c /W0 tsrc.c */
/* LINK tsrc tsrca; */
/*-----*/
/* Call : TSRC [/tkeycode|/tnumber...] */
*****/

```

/*== Add include files =====*/

```

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <bios.h>

```

/*== Type definitions =====*/

```

typedef unsigned char BYTE; /* Create a byte */
typedef unsigned int WORD;

```

```

typedef BYTE BOOL; /* Like BOOLEAN in Pascal */
typedef union vel far * VP; /* VP is a FAR pointer to video RAM */

typedef void (*OAFP)(void); /* Pointer to function without arguments */
typedef void (*SHKFP)( WORD KeyMask, BYTE ScCode ); /* TsrSetHotkey */

/*== Macros =====*/
#ifdef MK_FP /* MK_FP no defined yet? */
#define MK_FP(seg, ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
#endif
#define VOFS(x,y) ( 80 * ( y ) + ( x ) )
#define VPOS(x,y) (VP) ( vptr + VOFS( x, y ) )

/*== Structures and unions =====*/
struct velb { /* Describes a screen position as 2 bytes */
    BYTE thechar, /* ASCII code */
    attribute; /* Corresponding attribute */
};

struct velw { /* Describes a screen position as 1 word */
    WORD content; /* Get ASCII character and attribute */
};

union vel { /* Describes a screen position */
    struct velb h;
    struct velw x;
};

/*== Add functions from the assembler module =====*/

```

```

extern void TsrInit( BOOL tc, void (*fct)(void), unsigned heap );
extern BOOL TsrIsInst( BYTE i2F_fctnr );
extern void TsrUnInst( void );
extern OAFP TsrSetPtr( void far *fct );
extern BOOL TsrCanUnInst( void );
extern void TsrCall( void );
extern void far TsrSetHotkey( WORD keymask, BYTE sccode );

/*== Constants and macros =====*/

#ifdef __TURBOC__                                /* Compiling with Turbo C? */
    #include <alloc.h>
    #define TC TRUE                                /* Yes */
    #define KeyAvail() ( bioskey(1) != 0 )
    #define GetKey() bioskey(0)
#else                                              /* Compiling with Microsoft C */
    #include <malloc.h>
    #define TC FALSE
    #define KeyAvail() ( _bios_keybrd( _KEYBRD_READY ) != 0 )
    #define GetKey() _bios_keybrd( _KEYBRD_READ )
#endif

/*-- Scan codes of different keys -----*/

#define SC_ESC                0x01
#define SC_1                   0x02
#define SC_2                   0x03
#define SC_3                   0x04
#define SC_4                   0x05
#define SC_5                   0x06
#define SC_6                   0x07
#define SC_7                   0x08

```

#define SC_8	0x09
#define SC_9	0x0A
#define SC_0	0x0B
#define SC_HYPHEN	0x0C
#define SC_EQUALS	0x0D
#define SC_BACKSPACE	0x0E
#define SC_TAB	0x0F
#define SC_Q	0x10
#define SC_W	0x11
#define SC_E	0x12
#define SC_R	0x13
#define SC_T	0x14
#define SC_Y	0x15
#define SC_U	0x16
#define SC_I	0x17
#define SC_O	0x18
#define SC_P	0x19
#define SC_LBRACKET	0x1A
#define SC_RBRACKET	0x1B
#define SC_ENTER	0x1C
#define SC_CONTROL	0x1D
#define SC_A	0x1E
#define SC_S	0x1F
#define SC_D	0x20
#define SC_F	0x21
#define SC_G	0x22
#define SC_H	0x23
#define SC_J	0x24
#define SC_K	0x25
#define SC_L	0x26
#define SC_SEMICOLON	0x27
#define SC_APOSTROPHE	0x28

#define SC_GRAVE	0x29
#define SC_SHIFT_LEFT	0x2A
#define SC_BACKSLASH	0x2B
#define SC_Z	0x2C
#define SC_X	0x2D
#define SC_C	0x2E
#define SC_V	0x2F
#define SC_B	0x30
#define SC_N	0x31
#define SC_M	0x32
#define SC_COMMA	0x33
#define SC_PERIOD	0x34
#define SC_SLASH	0x35
#define SC_SHIFT_RIGHT	0x36
#define SC_ASTERISK	0x37
#define SC_ALT	0x38
#define SC_SPACE	0x39
#define SC_CAPS	0x3A
#define SC_F1	0x3B
#define SC_F2	0x3C
#define SC_F3	0x3D
#define SC_F4	0x3E
#define SC_F5	0x3F
#define SC_F6	0x40
#define SC_F7	0x41
#define SC_F8	0x42
#define SC_F9	0x43
#define SC_F10	0x44
#define SC_NUM_LOCK	0x45
#define SC_SCROLL_LOCK	0x46
#define SC_CURSOR_HOME	0x47
#define SC_CURSOR_UP	0x48

```
#define SC_CURSOR_PG_UP      0x49
#define SC_NUM_MINUS         0x4A
#define SC_CURSOR_LEFT      0x4B
#define SC_NUM_5             0x4C
#define SC_CURSOR_RIGHT     0x4D
#define SC_NUM_PLUS         0x4E
#define SC_CURSOR_END       0x4F
#define SC_CURSOR_DOWN      0x50
#define SC_CURSOR_PG_DOWN   0x51
#define SC_INSERT            0x52
#define SC_DELETE            0x53
#define SC_SYS_REQUEST       0x54
#define SC_F11               0x57
#define SC_F12               0x58
#define SC_NOKEY             0x80
/* No more keys */
```

```
/*-- Bit masks for the different toggle keys -----*/
```

```
#define RSHIFT      1          /* Right SHIFT key */
#define LSHIFT      2          /* Left SHIFT key */
#define CTRL        4          /* CTRL key */
#define ALT          8          /* ALT key */
#define SYSREQ      1024       /* SYS-REQ key (AT keyboard only) */
#define BREAK       4096       /* BREAK key */
#define NUM         8192       /* NUM LOCK key */
#define CAPS        16384      /* CAPS LOCK key */
#define INSERT      32768      /* INSERT key */
```

```
#define I2F_CODE 0xC4 /* Function number INT 2F */
#define I2F_FCT_0 0xAA /* Code for INT 2F, function 0 */
#define I2F_FCT_1 0xBB /* Code for INT 2F, function 1 */
```

```

#define NOF          0x07                      /* Normal color */
#define INV          0x70                      /* Inverse color */
#define HNOF         0x0f                      /* Bright normal color */
#define HINV         0xf0                      /* Bright inverse color */

#define HEAP_FREE 1024                        /* Allocate 1K of heap space */

#define TRUE  ( 0 == 0 )                      /* Constants used with BOOL */
#define FALSE ( 0 == 1 )

/*== Global variables =====*/
VP vpptr;                /* Pointer to the first character in video RAM */
unsigned atimes = 0;      /* Number of TSR activations */
union vel * scrbuf;       /* Pointer to a screen buffer */
char * blkspc;            /* Pointer to a blank line */

/*****
*  Function          : D I S P _ I N I T
*  -----
*  Task              : Passes the base address to video RAM.
*  Input parameters : None
*  Return values     : None
*****/

void disp_init(void)
{
    union REGS regs;      /* Processor registers for interrupt call */

    regs.h.ah = 15;        /* Function number: Get video mode */
    int86(0x10, &regs, &regs); /* Call BIOS video interrupt */

```

```

/*-- Compute base address of video RAM -----*/

vptr = (VP) MK_FP((regs.h.al == 7) ? 0xb000 : 0xb800, 0);
}

/*****
*   Function           : D I S P _ P R I N T
**-----**
*   Task               : Displays a string on the screen.
*   Input parameters   : - COLUMN   = Display column
*                       - SCROW     = Display row
*                       - DCOLR    = Character attribute
*                       - STRING    = Pointer to a string
*   Return values      : None
*****/

void disp_print(BYTE column, BYTE scrow, BYTE dcolr, char * string)
{
    register VP lptr;          /* Floating pointer for video RAM access */

    lptr = VPOS(column, scrow);      /* Set pointer in video RAM */
    for ( ; *string ; ++lptr)        /* Execute string */
    {
        lptr->h.thechar = *(string++); /* Write characters to video RAM */
        lptr->h.attribute = dcolr;     /* Set character attribute */
    }
}

/*****
*   Function           : S A V E _ S C R E N
**-----**
*   Task               : Saves the screen contents to a buffer.
**

```



```

*   Input parameters : - SPTR      = Pointer to the buffer in which the *
*                           screen contents will be saved.                *
*   Return values    : None                                              *
*   Info             : The buffer must be large enough to                *
*                           store the screen contents.                    *
*****/

```

```

void save_screen( union vel * sptr )
{
    register VP lptr;          /* Floating pointer for video RAM access */
    unsigned i;                /* Loop counter */
    lptr = VPOS(0, 0);         /* Set pointer in video RAM */

    for (i=0; i<2000; i++)     /* Execute the 2000 screen positions */
        (sptr++)->x.content = (lptr++)->x.content; /* Store char. & attb. */
}

```

```

/*****
*   Function          : R E S T O R E _ S C R E E N                      *
**-----**
*   Task             : Copies the contents of a buffer to video RAM.    *
*   Input parameters : - SPTR      = Pointer to the buffer whose contents *
*                           are to be copied to video RAM                *
*   Return values     : None                                              *
*****/

```

```

void restore_screen( union vel * sptr )
{
    register VP lptr;          /* Floating pointer for video RAM access */
    unsigned i;                /* Loop counter */
    lptr = VPOS(0, 0);         /* Set pointer in video RAM */
}

```

```

for (i=0; i<2000; i++)          /* Execute the 2000 screen positions */
    (lptr++)->x.content = (sptr++)->x.content; /* Return ch. & attb. */
}

```

```

/*****
*   Function           : E N D F C T                               *
**-----**
*   Task               : Called when the TSR program is uninstalled. *
*   Input parameters   : None                                         *
*   Return values      : None                                         *
*   Info               : This procedure must be FAR, so that it can be *
*                       called by the installed copy of the TSR.      *
*****/

```

```

void far endFCT( void )
{
    /*-- Release the allocated buffers -----*/

    free( blnkspace );          /* Release allocated buffer */
    free( (void *) scrbuf );    /* Release allocated buffer */

    printf("The TSR was called %u times.\n", atimes);
}

```

```

/*****
*   Function           : T S R                                       *
**-----**
*   Task               : Called by the assembler module after the hotkey *
*                       is pressed.                                     *
*   Input parameters   : None                                         *
*   Return values      : None                                         *
*****/

```

```

void tsr( void )
{
    BYTE i;                                /* Loop counter */

    ++atimes;                             /* Increment call counter */

    while ( KeyAvail() )                   /* Clear keyboard buffer */
        GetKey();

    disp_init();                           /* Get video RAM address */
    save_screen( scrbuf );                 /* Store current screen contents */
    for (i=0; i<25; i++)                   /* Execute 25 screen rows */
        disp_print(0, i, INV, blnkspace); /* Clear row */
    disp_print(34, 11, INV, "My first TSR.");
    disp_print(29, 13, INV, "Please press a key ...");
    GetKey();                              /* Wait for a key */
    restore_screen( scrbuf );              /* Restore old screen contents */
}

/*****
/* GetHeapEnd: Gets the current end of heap, adapting to compilers. */
/* Input      : None */
/* Output     : Pointer to the first byte after the end of heap */
*****/

void far *GetHeapEnd( void )
{
    #ifdef __TURBOC__                      /* Turbo C? */
        return (void far *) sbrk(0);
    #else                                  /* No --> MSC */
        struct _heapinfo hi;              /* Structure with heap entry information */

```

```

unsigned heapstatus;                                /* Status of _heapwalk() */
void far *lastblk;                                  /* Pointer to last block allocated */

hi._pentry = NULL;                                  /* Start at start of heap */

/*-- Execute heap up to last block -----*/

while( (heapstatus = _heapwalk( &hi )) != _HEAPEND )
    if ( hi._useflag == _USEDENTRY )                /* Block used? */
        lastblk = (void far *) ((BYTE far *) hi._pentry + hi._size + 1);

    return lastblk;
#endif
}

/*****
/* ParamGetHotKey: Checks command line parameters for the hotkey      */
/*                  switch (/T) and implements these keys.            */
/* Input   : ARGV,      = Switch parameters, similar to main()        */
/*          ARGV                                                */
/*          KEYMASK = Pointer to variable for storing the key mask     */
/*          SCCODE = Pointer to 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 key exists for /T, SC_NOKEY is placed in the    */
/*            appropriate variable.                                     */
*****/

BOOL ParamGetHotKey(int argc, char *argv[], WORD *KeyMask, BYTE *ScCode)

{

```

```

struct ToggleKey {
    char Name[7];
    WORD WVal;
};

static struct ToggleKey TogKeys[9] =
    { { "LSHIFT", LSHIFT },
      { "RSHIFT", RSHIFT },
      { "CTRL", CTRL },
      { "ALT", ALT },
      { "SYSREQ", SYSREQ },
      { "BREAK", BREAK },
      { "NUM", NUM },
      { "CAPS", CAPS },
      { "INSERT", INSERT }
    };

int i , j,
    code;
    char arg[80];

/* Loop counter */
/* Scan code conversion */
/* Argument access */

*KeyMask = 0;
*ScCode = SC_NOKEY;

for ( i = 1; i < argc; ++i )
    {
        strcpy( arg, argv[i] );
        strupr( arg );
        if ( arg[0] == '/' && arg[1] == 'T' )
            {
                /* /T argument found */
                code = atoi( &arg[2] );
                /* Convert code to binary */
                if ( code )
                    /* Conversion O.K.? */
            }
    }

```

```

    {
        if ( code < 128 )
            *ScCode = code;
        else
            return FALSE;
    }
else
    /* If not a number, must be the name of a toggle key */
    {
        for ( j = 0; j < 9; ++ j )
            if (!strcmp( TogKeys[j].Name, &arg[2] ))
                break;

        if ( j < 9 )
            *KeyMask = *KeyMask | TogKeys[j].WVal;
        else
            return FALSE;
    }
}

return TRUE;
}

/*****
**                               MAIN PROGRAM                               **
*****/

void main( int argc, char *argv[] )
{
    WORD KeyMask;
    BYTE ScCode;

    /* Get bit mask for toggle keys */
    /* Get scan codes of hotkeys */

    printf("TSRC - (c) 1988, 92 by Michael Tischer\n");
}

```

```

if ( !ParamGetHotKey( argc, argv, &KeyMask, &ScCode ))
{
    /* Error in command line parameters */
    printf( "Illegal command line parameters\n" );
    exit(1);
}

/*-- Command line parameters were O.K. -----*/

if ( !TsrIsInst( I2F_CODE ))          /* Program already installed? */
{
    /* No */
    atimes = 0;
    /* Program hasn't been enabled yet */
    printf( "TSR now installed. \n" );
    if ( KeyMask == 0 && ScCode == SC_NOKEY )    /* No parameters? */
    {
        /* No --> Default is ALT-H */
        TsrSetHotkey( ALT, SC_H );
        printf( "Press <ALT> + H to enable\n" );
    }
    else
        /* Install user-defined hotkeys */
        TsrSetHotkey( KeyMask, ScCode );

    /*-- Allocate buffer for screen management -----*/

    scrbuf = (union vel *) malloc(80 * 25 * sizeof(union vel));
    blknspace = (char *) malloc( 80 + 1 );
    /* Allocate buffer */
    *(blknspace + 80 ) = '\0';
    /* End buffer with NULL */
    memset(blknspace, ' ', 80);
    /* Fill buffer with spaces */

    TsrInit( TC, tsr, HEAP_FREE );
    /* Install program */
}
else
    /* Program already installed */
{
    /* Yes */
    if ( KeyMask == 0 && ScCode == SC_NOKEY )    /* No parameters? */

```

```

{
    if ( TsrCanUnInst() )
    {
        /* No --> Try uninstalling */
        (*(OAFP) TsrSetPtr(endFCT))();
        TsrUnInst();
        printf( "Program now uninstalled.\n" );
    }
    else
        printf( "Program cannot be uninstalled.\n" );
}
else
    /* Implement new hotkey */
    {
        printf( "New hotkey implemented\n" );
        (*(SHKFP) TsrSetPtr(TsrSetHotkey))( KeyMask, ScCode );
    }
}
}

```