27 TSR Programming

"Hard workers will become leaders."

TSR or "Terminate and Stay Resident" Programming is one of the interesting topics in DOS Programming. TSR programs are the one which seems to terminate, but remains resident in memory. So the resident program can be invoked at any time. Few TSR programs are written with the characteristic of TCR (Terminate Continue Running) i.e., TSR program seems to terminate, but continues to run in the background. TSR Programming is supposed to be an easy one, if you know the DOS internals. In this chapter, I have tried to explain the tough TSR Programming concept in a simpler manner.

27.1 DOS's non-reentrancy Problem

If a function can be called before it is finished, it is called reentrant. Unfortunately, DOS functions are non-reentrant. That is, we should not call a DOS function when it executes the same. Now, our intuition suggests us to avoid the DOS functions in TSR programs!

27.2 Switching Programs

As we know, DOS is not a multitasking operating system. So DOS is not meant for running two or more programs simultaneously! One of the major problems we face in TSR programming is that DOS's nature of switching programs. DOS handles switching programs, by simply saving the swapped-out program's complete register set and replacing it with the swapped-in program's registers. In DOS, if a program is put to sleep its registers are stored in an area called TCB (Task Control Block).

We must finish one process before another is undertaken. The main idea behind it is that, whenever we switch between programs, DOS switches our program's stack to its own internal set. And whatever that is pushed must be fully popped. For example, assume that we have a process currently running called *previous-process*, and we initiate another process in the meantime called *current-process*. In this case, the *current-process* will work fine, but when the *previous-process* just gets finished, it would find its stack data has been trashed by *current-process*. It is a serious injury! Everything will mess-up!

27.3 DOS Busy Flag

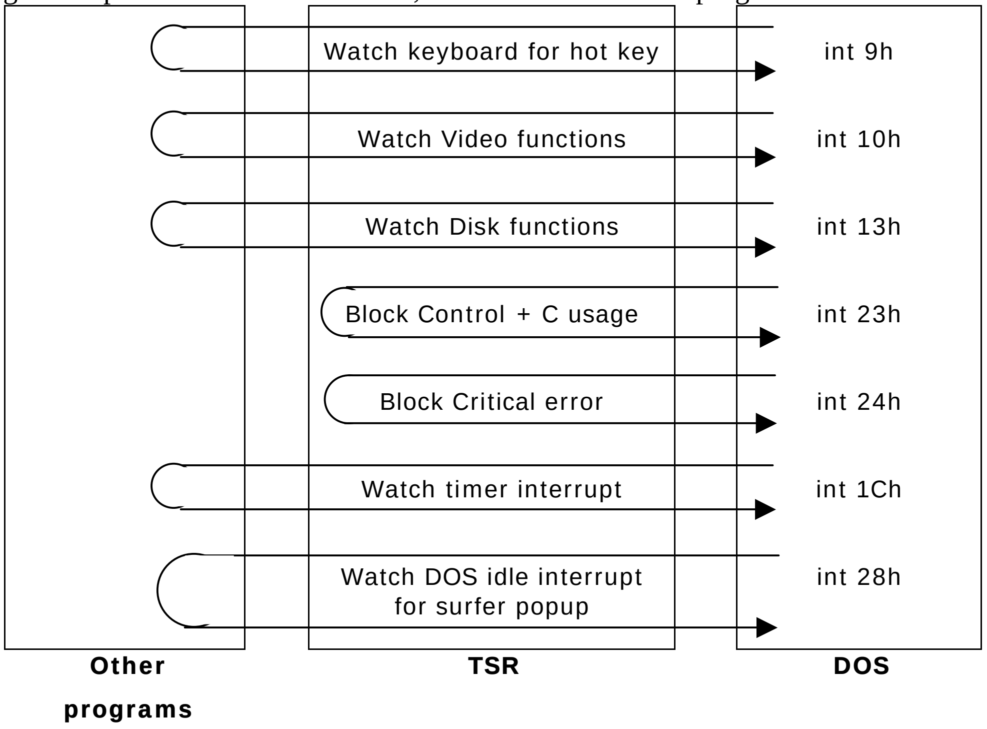
From the above discussion, we understand that before popping up our TSR program, we must check whether DOS is currently executing an internal routine (i.e., busy) or not. Surprisingly DOS also checks its status using a flag called "DOS Busy Flag". This "DOS Busy Flag" feature is undocumented and some programmers refer this flag as "DOS Critical Flag". We can also use this flag in our TSR program to check whether DOS is busy or not. For that, we have to use undocumented DOS function 34h.

27.4 BIOS Functions

As BIOS functions are reentrant, some programmers use BIOS functions in TSR programs. But professional programmers don't use BIOS functions, as the implementation of BIOS functions is quite different from machine to machine. In other words, BIOS is not compatible and there is no guarantee for its reentrancy. So for professional TSR programming, avoid BIOS functions too!

27.5 Popping up TSR

TSR programs can be made to reside in memory with the `keep( )` function. Then how does our TSR program understand, it is being requested by user? In other words, when to popup our TSR program? For that, we have to capture few interrupts. We have already seen that interrupt routines will be called whenever an interrupt is been generated. So if we replace the existing interrupt routine with our routine, we can make our TSR program "live".



Normally, TSR programmers capture Keyboard interrupt (int 9h), Control-C interrupt (int 23h), Control-break interrupt (int 1bh), Critical error interrupt (int 24h), BIOS disk interrupt (int

13h), Timer interrupt (int 1ch) and DOS Idle interrupt (int 28h). Indian TSR programmers often use int 8h as Timer interrupt. But other international TSR programmers use int 1ch as Timer interrupt.

The idea is that we have to block Control-C interrupt, Control-break interrupt and Critical error interrupt. Otherwise, there is a chance that the control will pass onto another program when our TSR program is in action. And it will spoil everything!

We must also monitor other interrupts—Keyboard interrupt, BIOS disk interrupt, Timer interrupt and DOS Idle interrupt, and we have to chain them. I hope by looking at the figure, you can understand the concept better.

27.6 IBM's Interrupt-Sharing Protocol

Almost all TSR utilities came with the property of unloading itself from the memory. But in order to unload the TSR, it must be the last TSR loaded. For example, if we run TSR utilities namely "X" and "Y", we can unload only the last TSR loaded i.e., "Y". The problem here is that of sharing of interrupts by TSR programs. IBM has suggested a protocol for sharing system interrupts. Even though, this protocol is meant for sharing hardware interrupts, it can be used for software interrupts too. It is especially useful for unloading TSR programs from memory, irrespective of its loading sequence. That is, if we follow this protocol standard, we can unload any TSR at any time!

So, in order to unload any TSR at any time, all the TSR programs must use this protocol. But unfortunately, TSR programmers don't use this standard. So I omit the discussion of this protocol. If you are very particular to know more about this protocol, checkout the **Intshare.doc** file found on CD.

27.7 Rules for TSR Programming

It is wise to consider the following rules, when you programming TSR:

1. Avoid DOS functions. If possible, avoid BIOS functions too!
2. When DOS busy flag is non-zero, DOS is executing interrupt 21h function. So we must wait and watch DOS busy flag.
3. When DOS is busy waiting for console input, we can disturb DOS regardless of the DOS busy flag setting. So you should watch interrupt 28h.
4. Use "signature" mechanism to check whether the TSR is already loaded or not. And so prevent multiple copies.
5. Our TSR program must use its own stack, and **not** that of the running process.
6. Other TSR programs might be chained to interrupts. So we must also chain any interrupt vector that our program needs.
7. TSR programs should be compiled in Small memory model.
8. *However* you may need to compile in compact , large or huge memory model if you use file operations with `getdta( )` and `setdta( )` functions.
9. TSR programs should be compiled with stack checking turned off.

27.8 TSR Template

Tom Grubbe has written a utility called PC-PILOT Programmer's Pop-Up. PC-PILOT is a good substitute for the commercial Sidekick utility. Full source code of PC-PILOT is available on the CD. Source codes of PC-PILOT run up to several pages and so I have avoided listing the codes here. However, I list the codes of **Tsr.c** file. This file can be treated as a good TSR Template and it reduces the pain of TSR programming.

```
/*
      TSR.C by Tom Grubbe
*/

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

#define TRUE      1
#define FALSE     0

/* --- vectors ---- */
#define DISK      0x13
#define INT28     0x28
#define KYBRD     0x9
#define CRIT      0x24
#define DOS       0x21
#define CTRLC     0x23
#define CTRLBRK   0x1b
#define TIMER     0x1c

typedef struct {
      int bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,fl;
} IREGS;
unsigned scancode;
unsigned keymask;

extern char signature[];
int unloading;                 /* TSR unload flag */

static void (*UserRtn)(void); /* Pointer to user's start routine */
static void (*InitRtn)(void); /* Pointer to user's initialization
                                 routine */

/* ---- interrupt vector chains ----- */
static void interrupt (*oldbreak)(void);
static void interrupt (*oldctrlc)(void);
static void interrupt (*oldtimer)(void);
static void interrupt (*old28)(void);
static void interrupt (*oldkb)(void);
static void interrupt (*olddisk)(void);
static void interrupt (*oldcrit)(void);

/* ------ ISRs fot the TSR --------- */
static void interrupt newtimer(void);
static void interrupt new28(void);
static void interrupt newkb(void);
static void interrupt newdisk(IREGS);
static void interrupt newcrit(IREGS);
static void interrupt newbreak(void);

static unsigned sizeprogram;  /* TSR's program size */
static unsigned dosseg;       /* DOS segment address */
static unsigned dosbusy;      /* offset to InDos flag */
static unsigned psps[2];      /* table of DOS PSP addresses */
static int pspctr;            /* # of DOS PSP addresses */
static int diskflag;          /* disk BIOS busy flag */
static unsigned mcbseg;       /* address of 1st DOS mcb */

static char far *mydta;       /* TSR's DTA  */
static unsigned myss;         /* TSR's stack segment */
static unsigned mysp;         /* TSR's stack pointer */
static unsigned intpsp;       /* Interrupted PSP address */
static int running;           /* TSR running indicator */
static int hotkey_flag;       /* Hotkey pressed flag */

/* ------ local prototypes ------- */
void tsr(void (*FPtr)(void), void (*InitFPtr)(void));

static void tsr_init(void);
static void resinit(void);
static void unload(void);
static void resterm(void);
static void pspaddr(void);
static void dores(void);
static void resident_psp(void);
static void interrupted_psp(void);
static int resident(char *signature);
static int test_hotkeys(int ky);

#define signon(s) printf("\n%s %s", signature, s);
```

```
void tsr(void (*FPtr)(void), void (*InitFPtr)(void))
{
      UserRtn = FPtr;
      InitRtn = InitFPtr;
      tsr_init();
      if (resident(signature) == FALSE)   {
            /* ------- initial load of TSR program -------- */
#ifdef DEBUG
            (*UserRtn)();
            return;
#else
            /* ------- Terminate and Stay Resident -------- */
            (*InitRtn)();     /* user's init function */
            resinit();
#endif
      }
      signon("is already installed.\n");
}

/* --------- initialize TSR control values ---------- */
static void tsr_init()
{
      unsigned es, bx;

      /* --------- get address of DOS busy flag --------- */
      _AH = 0x34;
      geninterrupt(DOS);
      dosseg  = _ES;
      dosbusy = _BX;
      /* --------- get the seg addr of 1st DOS MCB --------- */
      _AH = 0x52;
      geninterrupt(DOS);
      es = _ES;
      bx = _BX;
      mcbseg = peek(es, bx-2);
      /* --------- get address of resident program's dta -------- */
      mydta = getdta();
      /* --------- get address of PSP in DOS 2.x --------- */
      if (_osmajor < 3)
            pspaddr();
}

/* --------- establish & declare residency ---------- */
static void resinit()
{
      myss = _SS;
      mysp = _SP;
```

```
        oldtimer = getvect(TIMER);
        old28 = getvect(INT28);
        oldkb = getvect(KYBRD);
        olddisk = getvect(DISK);
        /* ------- attach vectors to resident program ------- */
        setvect(TIMER, newtimer);
        setvect(KYBRD, newkb);
        setvect(INT28, new28);
        setvect(DISK, newdisk);
        /* -------- compute program's size -------- */
        sizeprogram = myss + ((mysp+50) / 16) - _psp;
        /* -------- terminate and stay resident -------- */
        keep(0, sizeprogram);
}

/* --------- break handler ----------- */
static void interrupt newbreak()
{
        return;
}

/* ---------- critical error ISR --------- */
static void interrupt newcrit(IREGS ir)
{
        ir.ax = 0;        /* ignore critical errors */
}

/* -------- BIOS disk functions ISR --------- */
static void interrupt newdisk(IREGS ir)
{
        diskflag++;
        (*olddisk)();
        ir.ax = _AX;            /* for the register returns */
        ir.cx = _CX;
        ir.dx = _DX;
        ir.fl = _FLAGS;
        --diskflag;
}

/* -------- test for the hotkey --------- */
static int test_hotkeys(int ky)
{
        static unsigned biosshift;

        biosshift = peekb(0, 0x417);
        if (ky == scancode && (biosshift & keymask) == keymask)
              hotkey_flag = !running;
```

```
      return hotkey_flag;
}

/* --------- keyboard ISR ---------- */
static void interrupt newkb()
{
      static int kbval;

      if (test_hotkeys(inportb(0x60)))    {
            /* reset the keyboard */
            kbval = inportb(0x61);
            outportb(0x61, kbval | 0x80);
            outportb(0x61, kbval);
            outportb(0x20, 0x20);
      }
      else
            (*oldkb)();
}

/* --------- timer ISR ---------- */
static void interrupt newtimer()
{
      (*oldtimer)();
      test_hotkeys(0);
      if (hotkey_flag && peekb(dosseg, dosbusy) == 0) {
            if (diskflag == 0)      {
                  outportb(0x20, 0x20);
                  hotkey_flag = FALSE;
                  dores();
            }
      }
}

/* ---------- 0x28 ISR ---------- */
static void interrupt new28()
{
      (*old28)();
      if (hotkey_flag && peekb(dosseg, dosbusy) != 0) {
            hotkey_flag = FALSE;
            dores();
      }
}

/* ------ switch psp context from interrupted to TSR ------ */
static void resident_psp()
{
      int pp;
      if (_osmajor < 3) {
            /* --- save interrupted program's psp (DOS 2.x) ---- */
            intpsp = peek(dosseg, *psps);
            /* ------- set resident program's psp ------- */
            for (pp = 0; pp < pspctr; pp++)
                  poke(dosseg, psps[pp], _psp);
      }
      else  {
            /* ----- save interrupted program's psp ------ */
            intpsp = getpsp();
            /* ------ set resident program's psp ------- */
            _AH = 0x50;
            _BX = _psp;
            geninterrupt(DOS);
      }
}

/* -------- switch psp context from TSR to interrupted --------- */
static void interrupted_psp()
{
      int  pp;

      if (_osmajor < 3) {
            /* --- reset interrupted psp (DOS 2.x) ---- */
            for (pp = 0; pp < pspctr; pp++)
                  poke(dosseg, psps[pp], intpsp);
      }
      else  {
            /* ------ reset interrupted psp ------- */
            _AH = 0x50;
            _BX = intpsp;
            geninterrupt(DOS);
      }
}

/* -------- execute the resident program ---------- */
static void dores()
{
      static char far *intdta;      /* interrupted DTA */
      static unsigned intsp;        /*   "      stack pointer */
      static unsigned intss;        /* "      stack segment */
      static unsigned ctrl_break;   /* Ctrl-Break setting */

      running = TRUE;          /* set TSR running metaphore */
      disable();
      intsp = _SP;
      intss = _SS;
      _SP = mysp;
      _SS = myss;
      oldcrit = getvect(CRIT);
      oldbreak = getvect(CTRLBRK);
      oldctrlc = getvect(CTRLC);
      setvect(CRIT, newcrit);
      setvect(CTRLBRK, newbreak);
      setvect(CTRLC, newbreak);
      ctrl_break = getcbrk();       /* get ctrl break setting */
      setcbrk(0);                   /* turn off ctrl break logic */
      intdta = getdta();            /* get interrupted dta */
      setdta(mydta);                /* set resident dta */
      resident_psp();               /* swap psps */
      enable();

      (*UserRtn)();                 /* call the TSR program here */

      disable();
      interrupted_psp();            /* reset interrupted psp */
      setdta(intdta);               /* reset interrupted dta */
      setvect(CRIT, oldcrit);       /* reset critical error */
      setvect(CTRLBRK, oldbreak);
      setvect(CTRLC, oldctrlc);
      setcbrk(ctrl_break);          /* reset ctrl break */
      _SP = intsp;                  /* reset interrupted stack */
      _SS = intss;
      enable();
      if (unloading)
            unload();
      running = FALSE;
}

/* ------ test to see if the program is already resident -------- */
static int resident(char *signature)
{
      char *sg;
      unsigned df;
      unsigned blkseg, mcbs = mcbseg;

      df = _DS - _psp;
      /* --- walk through mcb chain & search for TSR --- */
      while (peekb(mcbs, 0) == 0x4d)      {
            blkseg = peek(mcbs, 1);
            if (peek(blkseg, 0) == 0x20cd)      {
                  /* ---- this is a psp ---- */
                  if (blkseg == _psp)
                        break;            /* if the transient copy */
```

```
                    for (sg = signature; *sg; sg++)
                          if (*sg != peekb(blkseg+df, (unsigned)sg))
                                break;
                          if (*sg == '\0') /*- TSR is already resident -*/
                                return TRUE;
              }
              mcbs += peek(mcbs, 3) + 1;
        }
        return FALSE;
  }

  /* --------- find address of PSP (DOS 2.x) ----------- */
  static void pspaddr()
  {
        unsigned adr = 0;

        disable();
        /* ------- search for matches on the psp in dos -------- */
        while (pspctr < 2 &&
                    (unsigned)((dosseg<<4) + adr) < (mcbseg<<4))    {
              if (peek(dosseg, adr) == _psp)      {
                    /* ------ matches psp, set phoney psp ------- */
                    _AH = 0x50;
                    _BX = _psp + 1;
                    geninterrupt(DOS);
                    /* ---- did matched psp change to the phoney? ----- */
                    if (peek(dosseg, adr) == _psp + 1)
                          /*---- this is a DOS 2.x psp placeholder ----*/
                          psps[pspctr++] = adr;
                    /* ----- reset the original psp ------ */
                    _AH = 0x50;
                    _BX = _psp;
                    geninterrupt(DOS);
              }
              adr++;
        }
        enable();
  }

  /* -------- unload the rsident program --------- */
  static void unload()
  {
        if (getvect(DISK) == (void interrupt (*)()) newdisk)
              if (getvect(KYBRD) == newkb)
                    if (getvect(INT28) == new28)
                          if (getvect(TIMER) == newtimer)     {
                                resterm();
```

```
                              return;
                        }
      /* --- another TSR is above us, cannot unload --- */
      putch(7);
}

/* --------- TSR unload function ----------- */
static void resterm()
{
      unsigned mcbs = mcbseg;
      /* restore the interrupted vectors */
      setvect(TIMER, oldtimer);
      setvect(KYBRD, oldkb);
      setvect(INT28, old28);
      setvect(DISK, olddisk);
      /* obliterate the signature */
      *signature = '\0';
      /* walk through mcb chain &
            release memory owned by the TSR */
      while (peekb(mcbs, 0) == 0x4d)      {
            if (peek(mcbs, 1) == _psp)
                  freemem(mcbs+1);
            mcbs += peek(mcbs, 3) + 1;
      }
}
```

27.9 PC-PILOT

In the last section we have seen the TSR Template that will be very useful for writing any TSR software. In this section, I just present the main program only. You can see how the TSR template (Tsr.c) is used in Pcpilot main program.

```
/*
      PCPILOT.C - This is the main( ) module for PCPILOT.EXE.
      It should be compiled in the small or tiny memory model.
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <scr.h>
#include <kbd.h>

int BorderClr = 0x09;
int TitleClr = 0x0c;
int TextClr = 0x0f;
int FooterClr = 0x0b;
int HighlightClr = 0x4f;
int Code = 0;                                   /* For Ascii()       */
int BoxIdx = 0;                                 /* For BoxCodes()    */
int ClrIdx = 0x00;                              /* For ColorCodes()  */
unsigned long NumIdx = 0L;                      /* For BaseConvert() */
int Row = 0, Col = 0;                           /* For Ruler()       */

void PcPilot(void);
void Initialize(void);
static int videomode(void);
void TitleScreen(void);

/*  #define DEBUG  */
char signature[] = "PC-PILOT";
extern unsigned _heaplen = 12288;
extern unsigned _stklen = 1024;
extern unsigned scancode[], keymask[];
extern int unloading;                           /* To UnInstall TSR */
void main(int argc, char *argv[])
{
      while (--argc > 0)      {
            ++argv;
            if (**argv != '-')
                  break;
            if (tolower(argv[0][1]) == 'x')     {
                  Initialize();
                  PcPilot();
                  return;
            }
      }
      Initialize();
      *scancode= 76;                      /* Alt(8) - '5'(76) on the keypad */
      *keymask = 8;
      tsr(PcPilot, TitleScreen);
}
typedef struct {
      char *str;
      int y;
} MENU;
MENU m[] = {
      " Ascii Table     ",  8,
      " Box Characters ",  9,
      " Hex/Dec/Binary ", 10,
      " Keyboard Codes ", 11,
      " Ruler           ", 12,
      " Color Codes     ", 13,
      " Printer Setup   ", 14,
      " Uninstall      ", 15,
      " Exit           ", 16
};
int Idx = 0;
int K;
int oldx, oldy;
static void DrawMenu(void);
static void HighLight(int code);
static void ExecuteMenuOptions(int index);

void PcPilot()
{
      ScrGetCur(&oldx, &oldy, 0);
      HideCur();
      ScrPush();
      DrawMenu();

      for (;;)    {
            HighLight(1);
            switch (K = KbdGetC())  {
                  case UP:
                  case LEFT:
                        HighLight(0);
                        if (--Idx < 0) Idx = 8;
                        break;
                  case DN:
                  case RIGHT:
                        HighLight(0);
                        if (++Idx > 8) Idx = 0;
                        break;
                  case PGUP:
                  case HOME:
                        HighLight(0);
                        Idx = 0;
                        break;
                  case PGDN:
                  case END:
                        HighLight(0);
                        Idx = 8;
                        break;
                  case RET:
                        if (Idx == 7 || Idx == 8)     {
                              if (Idx == 7) unloading = 1;
                              ScrPop(1);
                              ScrSetCur(oldx, oldy, 0);
                              return;
                        }
                        if (Idx == 4)     {
                              ScrPop(1);
                              Ruler();
                              ScrPush();
                              DrawMenu();
                        }
                        ExecuteMenuOptions(Idx);
                        break;
                  case ESC:
                        ScrPop(1);
                        ScrSetCur(oldx, oldy, 0);
                        return;
                  default:
                        if ((K = K&0x00ff) != 0)      {
                              if (!strchr("abhkrcpue", tolower(K)))
                                                break;
                              HighLight(0);
                              switch (tolower(K))     {
                                    case 'a': Idx = 0; break;
                                    case 'b': Idx = 1; break;
                                    case 'h': Idx = 2; break;
                                    case 'k': Idx = 3; break;
                                    case 'r': Idx = 4;
                                          ScrPop(1);
                                          Ruler();
                                          ScrPush();
                                          DrawMenu();
                                          break;
                                    case 'c': Idx = 5; break;
                                    case 'p': Idx = 6; break;
                                    case 'u': Idx = 7;
                                          unloading = 1;
                                    case 'e': Idx = 8;
                                                  ScrPop(1);
                                          ScrSetCur(oldx, oldy, 0);
                                                  return;
                                    default : continue;
                              }
                              HighLight(1);
                              ExecuteMenuOptions(Idx);
                        }
                        break;
            }
      }
}
```

```
static void DrawMenu()
{
      register int i;

      ShadowBox(31,5,48,19, 2, BorderClr);
      PutStr(32,6, TitleClr, "   PC - PILOT   ");
      PutStr(31,7, BorderClr,"╞════════════════╡");
      PutStr(31,17,BorderClr,"╞════════════════╡");
      PutStr(32,18,FooterClr," %c %c <Esc> exits", 24,25);

      for (i=0; i<9; i++)     {
            PutStr(32,8+i, TextClr, "%s", m[i].str);
            PutStr(33,8+i, FooterClr, "%c", m[i].str[1]);
      }
      HighLight(1);
}

static void HighLight(int code)
{
      switch (code)     {
            case 0:
                  PutStr(32,m[Idx].y, TextClr, "%s", m[Idx].str);
                  PutStr(33,m[Idx].y, FooterClr, "%c", m[Idx].str[1]);
                  break;
            case 1:
                 PutStr(32,m[Idx].y, ~TextClr & 0x7f, "%s", m[Idx].str);
            PutStr(33,m[Idx].y, ~FooterClr & 0x7f, "%c", m[Idx].str[1]);
                  break;
      }
}

static void ExecuteMenuOptions(int index)
{
      switch (index)    {
            case 0:  Ascii(); return;
            case 1:  BoxCodes(); return;
            case 2:  BaseConvert(); return;
            case 3:  KeyCodes(); return;
            case 4:  return;
            case 5:  ColorCodes(); return;
            case 6:  PrintCodes(); return;
            case 7:  return;
      }
}
```

```
static void Initialize()
{
      int vmode;

      vmode = videomode();
      if ((vmode != 2) && (vmode != 3) && (vmode != 7))     {
            printf("Must be in 80 column text mode.\n");
            exit(1);
      }
      InitScr();
      if (VideoMode == MONO)  {
            BorderClr     = 0x0f;
            TitleClr      = 0x0f;
            TextClr       = 0x07;
            FooterClr     = 0x0f;
            HighlightClr  = 0x70;
      }
}

static int videomode()
{
      union REGS r;

      r.h.ah = 15;
      return int86(0x10, &r, &r) & 255;
}

static void TitleScreen()
{
      Cls();
      ShadowBox(18,8,59,16, 2, BorderClr);
      PutStr(19,9, TextClr, "               PC - PILOT               ");
      PutStr(19,10,FooterClr,"          Programmer's Pop-Up          ");
      PutStr(19,12, TextClr,"     FREEware written by Tom Grubbe     ");
      PutStr(19,13, TextClr," Released to the Public Domain 01-12-90 ");
      PutStr(19,15, TextClr,"          Alt-5 (keypad)                ");
      PutStr(23,15, TitleClr,  "Press");
      PutStr(44,15, TitleClr,  "To Activate");
      ScrSetCur(0,18,0);
}
```

Suggested Projects

1. Write a Screen Thief utility. The Screen Thief will capture the screen, when a hot-key is pressed. Depending upon the mode you set, when you load the TSR, Screen Thief will store the screen into BMP or GIF or JPEG.