```
:::::::::::::: atm.sh ::::::::::::
#!/bin/sh
#
# script: atm.sh
# purpose: simulate an tmp with the option to 'play again'
#
while true
do
        echo "How much do you want? "
        read amount
        echo "here is $amount dollars"
        printf "\a"
        if play_again0
        then
                :
        else
                break
        fi
done

:::::::::::::: play_again1.c ::::::::::::::

#include       <stdio.h>
#include       <termios.h>
/*
 * play_again1.c
 *
 *      purpose: ask if user wants another transaction
 *       method: set tty into char-by-char mode, read char, return result
 *      returns: 0=>yes, 1=>no
 *       better: do no echo inappropriate input
 */
#define QUESTION       "Do you want another transaction"
main()
{
        int     response;
        set_crmode();                           /* set chr-by-chr mode */
        response = get_response(QUESTION);      /* get some answer     */
        reset_tty_mode();                       /* restore tty state   */
        return response;
}
get_response( char *question )
/*
 * purpose: ask a question and wait for a y/n answer
 *  method: use getchar and complain about non-y/n input
 * returns: 0=>yes, 1=>no
 */
{
        int     input;
        printf("%s (y/n)?", question);
        while ( (input = getchar() ) != EOF ) {
                if ( input == 'y' || input == 'Y' )
                        return 0;
                if ( input == 'n' || input == 'N' )
                        return 1;
                printf("\ncannot understand %c, Please type y or no\n",input);
        }
}

struct termios original_mode;         /* used by restore     */

set_crmode()
/*
 * purpose: put file descriptor 0 (i.e. stdin) into chr-by-chr mode
 *  method: use bits in termios
 */
{
        struct  termios ttystate;
        tcgetattr( 0, &ttystate);             /* read curr. setting  */
        original_mode          = ttystate;    /* remember these      */
        ttystate.c_lflag        &= ~ICANON;   /* no buffering                */
        ttystate.c_cc[VMIN]     =  1;         /* get 1 char at a time        */
        tcsetattr( 0 , TCSANOW, &ttystate);   /* install settings    */
}
reset_tty_mode()
{
        tcsetattr(0, TCSANOW, &original_mode);       /* put back orig mode  */
}
```

```
#include       <stdio.h>
#include       <termios.h>
#include       <fcntl.h>
#include       <signal.h>

/*
 * play_again4.c
 *
 *      purpose: ask if user wants another transaction
 *       method: set tty into chr-by-chr, no-echo mode
 *               set tty into no-delay mode
 *               read char, return result
 *               resets terminal modes on SIGINT, ignores SIGQUIT
 *      returns: 0=>yes, 1=>no, 2=>timeout
 *       better: reset terminal mode on Interrupt
 */
#define ASK            "Do you want another transaction"
#define TIMEOUT        5       /* in seconds  */

main()
{
        int     response;
        int     ctrl_c_handler();

        set_cr_noecho_mode();                   /* set -icanon, -echo  */
        set_nodelay_mode();                     /* noinput => EOF      */
        signal( SIGINT, ctrl_c_handler );       /* handle INT          */
        signal( SIGQUIT, SIG_IGN );             /* ignore QUIT signals */
        response = get_response(ASK, TIMEOUT);       /* get some answer     */
        reset_tty_mode();                       /* restore tty state   */
        return response;
}
get_response( char *question , int timeout )
/*
 * purpose: ask a question and wait for a y/n answer or timeout
 *  method: use getchar and complain about non-y/n input
 * returns: 0=>yes, 1=>no
 */
{
        int     input;
        printf("%s (y/n)?", question);
        while ( 1 ){
                input = getchar();                      /* getachar    */
                if ( input == 'y' || input == 'Y' )     /* Y?          */
                        return 0;
                if ( input == 'n' || input == 'N' )     /* N?          */
                        return 1;
                if ( input == EOF ){                    /* EOF?        */
                        if ( timeout-- == 0 )           /* outatime?   */
                                return 2;               /* sayso       */
                        sleep(1);                       /* notyet:wait */
                        putchar('?');
                }
        }
}

struct termios original_mode;                   /* used by restore     */
int            original_flags;                          /* used to restore     */

set_cr_noecho_mode()
/*
 * purpose: put file descriptor 0 into chr-by-chr mode and noecho mode
 *  method: use bits in termios
 */
{
        struct  termios ttystate;

        tcgetattr( 0, &ttystate);               /* read curr. setting  */
        original_mode          = ttystate;      /* remember these      */
        ttystate.c_lflag       &= ~ICANON;      /* no buffering                */
        ttystate.c_lflag       &= ~ECHO;        /* no echo either      */
        ttystate.c_cc[VMIN]    =  1;            /* get 1 char at a time        */
        tcsetattr( 0 , TCSANOW, &ttystate);     /* install settings    */
}
```

```
set_nodelay_mode()
/*
 * purpose: put file descriptor 0 into no-delay mode
 *  method: use fcntl to set bits
 *   notes: tcsetattr() will do something similar, but it is complicated
 */
{
        int     termflags;

        termflags = fcntl(0, F_GETFL);              /* read curr. settings */
        original_flags = termflags;          /* remember these      */
        termflags |= O_NDELAY;               /* flip on nodelay bit */
        fcntl(0, F_SETFL, termflags);        /* and install 'em     */
}

reset_tty_mode()
{
        tcsetattr(0, TCSANOW, &original_mode);       /* undo -icanon, -echo */
        fcntl( 0, F_SETFL, original_flags);  /* undo NoDelay               */
}

ctrl_c_handler()
/*
 * purpose: called if SIGINT is detected
 *  action: reset tty and scram
 */
{
        reset_tty_mode();
        exit(3);
}
:::::::::::::: demo_signal.c ::::::::::::::

#include       <stdio.h>
#include       <signal.h>

/**
 **     signal sampler
 **
 **             short program to demonstrate how
 **             signals can kill a process, bounce off a process
 **             or be caught by a process
 **/

main()
{
        int     catcher();             /* a function to call on CtrlC */
        int     i;

        printf("Case 1: no special arrangements..");
        for (i=0;i<10;i++){
                putchar('*');fflush(stdout);
                sleep(1);
        }
        putchar('\n');

        signal(SIGINT, SIG_IGN);      /* ignore INTerrupts   */
        printf("Case 2: ignoring interrupts..");
        for (i=0;i<10;i++){
                putchar('*');fflush(stdout);
                sleep(1);
        }
        putchar('\n');

        signal(SIGINT, catcher);      /* handle interruptions       */
        printf("Case 3: catching interrupts..");
        for (i=0;i<10;i++){
                putchar('*');fflush(stdout);
                sleep(1);
        }
        putchar('\n');
}

catcher()
{
        printf(" Ouch! \n");
        system("/bin/who");
        /* signal( SIGINT, catcher ); */
}
```