```
:::::::::::::: listchars.c ::::::::::::::
#include       <stdio.h>

/*
 *   listchars.c
 *      purpose: list individually all the chars seen on input
 *       output: char and ascii code, one pair per line
 *        input: stdin, until the letter Q
 *        notes: usesful to show that buffering/editing exists
 */
main()
{
       int     c, n = 0;

       while( ( c = getchar()) != 'Q' )
               printf("char %3d is %c code %d\n", n++, c, c );
}

:::::::::::::: nobuf.c ::::::::::::::
#include       <stdio.h>
#include       <fcntl.h>

/*
 *   nobuff.c
 *     purpose: demonstrate use of fcntl to set a fd attribute
 *      action: opens a file, then uses fcntl to change attrib
 *              then writes some data
 *        value: see how fcntl works
 */
#define oops(s)  { perror(s); exit(1); }
#define DATALEN 10000
char   buf[DATALEN] ;

main()
{
       int     fd, attribs;

       if ( ( fd = open( "junkfile", O_WRONLY|O_CREAT, 0644 ) ) == -1 )
               oops( "open" );
       attribs = fcntl( fd, F_GETFL );                        /* get attribs */
       attribs = attribs|O_SYNC ;                     /* turn on SYNC */
       if ( fcntl(fd,F_SETFL,attribs) == -1 )                 /* set attribs */
               oops( "fcntl" );
       if ( write( fd, buf, DATALEN ) == -1 )
               oops( "write" );
       if ( close( fd ) == -1 )
               oops( "close" );
}

:::::::::::::: append0.c ::::::::::::::
#include       <stdio.h>
#include       <fcntl.h>

/*
 *   append0.c
 *     purpose: demonstrate risks of appending with lseek() and write()
 *      action: opens a file, then lseeks to end, then waits, then writes
 *        value: run two of these and watch the fun
 *        usage: append0 filename message [delay]
 */
#define oops(s)  { perror(s); exit(1); }

main(int ac, char *av[])
{
       int     fd, delay = 1;

       if ( ac < 3 ) exit(2);                         /* usage       */
       if ( ac == 4 ) delay = atoi( av[3] );          /* delay arg   */

       if ( ( fd = open( av[1], O_WRONLY|O_CREAT, 0644 ) ) == -1 )
               oops( "open" );
       lseek( fd, 0L, 2 );                            /* seek to end */
       sleep(delay);                                  /* pause       */
       if ( write(fd,av[2],strlen(av[2])) == -1 )     /* write       */
               oops( "write" );
       write( fd, "\n", 1 );                          /* add newline */
       if ( close(fd) == -1 )
               oops( "close" );
}
```

```
:::::::::::::: append1.c ::::::::::::::
#include       <stdio.h>
#include       <fcntl.h>

/*
 *   append1.c
 *      purpose: demonstrate how to use O_APPEND to ensure appends
 *       action: opens a file, then lseeks to end, then waits, then writes
 *        value: run two of these and watch the fun
 *        usage: append1 filename message [delay]
 *        shows: how to set attributes when opening
 */

#define oops(s)  { perror(s); exit(1); }

main(int ac, char *av[])
{
        int     fd, delay = 1;

        if ( ac < 3 ) exit(2);                          /* usage       */
        if ( ac == 4 ) delay = atoi( av[3] );           /* delay arg   */

        if ( ( fd = open( av[1], O_WRONLY|O_CREAT|O_APPEND, 0644 ) ) == -1 )
                oops( "open" );
        lseek( fd, 0L, 2 );                             /* seek to end */
        sleep(delay);                                   /* pause       */
        if ( write(fd,av[2],strlen(av[2])) == -1 )      /* write       */
                oops( "write" );
        write( fd, "\n", 1 );                           /* add newline */
        if ( close(fd) == -1 )
                oops( "close" );
}
:::::::::::::: write0.c ::::::::::::::
#include       <stdio.h>
#include       <fcntl.h>

/*
 * write0.c
 *
 *      purpose: send messages to another terminal
 *       method: open the other terminal for output then
 *               copy from stdin to that terminal
 *        shows: a terminal is just a file supporting regular i/o
 *        usage: write0 ttyname
 */

main( int ac, char *av[] )
{
        int     fd;
        char    buf[BUFSIZ];

        /* check args */
        if ( ac != 2 ){
                fprintf(stderr,"usage: write0 ttyname\n");
                exit(1);
        }

        /* open devices */
        fd = open( av[1], O_WRONLY );
        if ( fd == -1 ){
                perror(av[1]); exit(1);
        }

        /* loop until EOF on input */
        while( fgets(buf, BUFSIZ, stdin) != NULL )
                if ( write(fd, buf, strlen(buf)) == -1 )
                        break;
        close( fd );
}
```

```
::::::::::::: write1.c :::::::::::::
#include       <stdio.h>
#include       <fcntl.h>
#include       <utmp.h>

/*
 * write1.c
 *
 *     purpose: send messages to another terminal
 *      method: open the other terminal for output then
 *              copy from stdin to that terminal
 *       usage: write1 username
 */

main( int ac, char *av[] )
{
        int     fd;
        char    buf[BUFSIZ];
        char    *get_tty(), *tty_for_user;

        /* check args */
        if ( ac != 2 ){
                fprintf(stderr,"usage: write0 logname\n");
                exit(1);
        }

        /* find user */
        tty_for_user = get_tty( av[1] );
        if ( tty_for_user == NULL )
                return 1;

        /* open device */
        sprintf(buf, "/dev/%s", tty_for_user);
        fd = open( buf, O_WRONLY );
        if ( fd == -1 ){
                perror(buf); exit(1);
        }

        /* loop until EOF on input */
        while( fgets(buf, BUFSIZ, stdin) != NULL )
                if ( write(fd, buf, strlen(buf)) == -1 )
                        break;
        close( fd );
}

char *
get_tty( char *logname )
/*
 * purpose: find the tty at which 'logname' is logged in
 * returns: a string or NULL if not logged in
 *  errors: does not handle multiple logins
 */
{
        static struct utmp utrec;
        int         utfd;
        int             namelen = sizeof( utrec.ut_name );
        char    *retval = NULL ;

        /* open file */
        if ( (utfd = open( UTMP_FILE, O_RDONLY )) == -1 )
                return NULL;

        /* look for a line where the user is logged in */
        while( read( utfd, &utrec, sizeof(utrec)) == sizeof(utrec) )
                if ( strncmp(logname, utrec.ut_name, namelen ) == 0 )
                {
                        retval = utrec.ut_line ;
                        break;
                }

        /* close file and return result */
        close(utfd);
        return retval;
}
```

```
:::::::::::::: echostate.c ::::::::::::::
#include       <stdio.h>
#include       <termios.h>

/*
 * echostate.c
 *   reports current state of echo bit in tty driver for fd 0
 *   shows how to read attributes from driver and test a bit
 */

main()
{
       struct termios info;

       tcgetattr( 0, &info );        /* read values from driver    */

       if ( info.c_lflag & ECHO )
              printf(" echo is on , since its bit is 1 \n");
       else
              printf(" echo if OFF, since its bit is 0\n");
}
:::::::::::::: setecho.c ::::::::::::::
#include       <stdio.h>
#include       <termios.h>

/*
 * setecho.c
 *   usage:  setecho [y|n]
 *   shows:  how to read, change, reset tty attributes
 */

main(int ac, char *av[])
{
       struct termios info;

       if ( ac == 1 ) exit(0);

       tcgetattr( 0, &info );               /* get attribs                */
       if ( av[1][0] == 'y' )
              info.c_lflag |= ECHO ;        /* turn on bit        */
       else
              info.c_lflag &= ~ECHO ;              /* turn off bit                */

       tcsetattr( 0, TCSANOW, &info );      /* set attribs        */
}
:::::::::::::: showtty.c ::::::::::::::
#include       <stdio.h>
#include       <termios.h>

/**
 **    showtty
 **    displays some of current tty settings
 **/

main()
{
       struct  termios ttyinfo;      /* this struct holds tty info */

       if ( tcgetattr( 0 , &ttyinfo ) == -1 ){   /* get info */
              perror( "cannot get params about stdin");
              exit(1);
       }
                                            /* show info */
       showbaud ( cfgetospeed( &ttyinfo ) ); /* get + show baud rate       */
       printf("The erase character is ascii %d, Ctrl-%c\n",
                     ttyinfo.c_cc[VERASE], ttyinfo.c_cc[VERASE]-1+'A');
       printf("The line kill character is ascii %d, Ctrl-%c\n",
                     ttyinfo.c_cc[VKILL], ttyinfo.c_cc[VKILL]-1+'A');

       show_some_flags( &ttyinfo );         /* show misc. flags    */
}
```

```
showbaud( thespeed )
/*
 *      prints the speed in english
 */
{
        printf("the baud rate is ");
        switch ( thespeed ){
                case B300:      printf("300\n");        break;
                case B600:      printf("600\n");        break;
                case B1200:     printf("1200\n");       break;
                case B1800:     printf("1800\n");       break;
                case B2400:     printf("2400\n");       break;
                case B4800:     printf("4800\n");       break;
                case B9600:     printf("9600\n");       break;
                default:        printf("Fast\n");       break;
        }
}

struct flaginfo { int  fl_value; char *fl_name; };

struct flaginfo input_flags[] = {

                IGNBRK  ,       "Ignore break condition",
                BRKINT  ,       "Signal interrupt on break",
                IGNPAR  ,       "Ignore chars with parity errors",
                PARMRK  ,       "Mark parity errors",
                INPCK   ,       "Enable input parity check",
                ISTRIP  ,       "Strip character",
                INLCR   ,       "Map NL to CR on input",
                IGNCR   ,       "Ignore CR",
                ICRNL   ,       "Map CR to NL on input",
                IXON    ,       "Enable start/stop output control",
                /* _IXANY  ,    "enable any char to restart output",  */
                IXOFF   ,       "Enable start/stop input control",
                0       ,       NULL };

struct flaginfo local_flags[] = {
                ISIG    ,       "Enable signals",
                ICANON  ,       "Canonical input (erase and kill)",
                /* _XCASE       ,       "Canonical upper/lower appearance", */
                ECHO    ,       "Enable echo",
                ECHOE   ,       "Echo ERASE as BS-SPACE-BS",
                ECHOK   ,       "Echo KILL by starting new line",
                0       ,       NULL };

show_some_flags( struct termios *ttyp )
/*
 *      show the values of two of the flag sets_: c_iflag and c_lflag
 *      adding c_oflag and c_cflag is pretty routine - just add new
 *      tables above and a bit more code below.
 */
{
        show_flagset( ttyp->c_iflag, input_flags );
        show_flagset( ttyp->c_lflag, local_flags );
}

show_flagset( int thevalue, struct flaginfo thebitnames[] )
/*
 * check each bit pattern and display descriptive title
 */
{
        int     i;

        for ( i=0; thebitnames[i].fl_value ; i++ ) {
                printf( "  %s is ", thebitnames[i].fl_name);
                if ( thevalue & thebitnames[i].fl_value )
                        printf("ON\n");
                else
                        printf("OFF\n");
        }
}
```