```
:::::::::::::: hello1.c ::::::::::::::
#include       <stdio.h>
#include       <curses.h>

/*
 * hello1.c
 *      purpose show the minimal calls needed to use curses
 *      outline initialize the functions, clear, move, draw, refresh, end
 */

main()
{
        /* part1 : set up package for terminal */
                initscr() ;

        /* part2 : do some screen operations  */
                clear();
                move(10,20);                            /* row10,col20 */
                addstr("Hello, world");                         /* add a string       */
                move(LINES-1,0);                        /* move to LL  */

        /* part3 : push requests to screen    */
                refresh();
                getch();

        /* part4 : wrapup                     */
                endwin();
}
:::::::::::::: hello2.c ::::::::::::::
#include       <stdio.h>
#include       <curses.h>

/*
 * hello2.c
 *      purpose show how to use curses functions with a loop
 *      outline initialize, draw stuff, wrap up
 */

main()
{
        int     i;

        initscr();

        /* part2 : do some screen operations  */
                clear();
                for(i=0; i<LINES; i++ ){
                        move( i, i+i );
                        if ( i%2 == 1 ) standout();
                        addstr("Hello, world");
                        if ( i%2 == 1 ) standend();
                }

        /* part3 : push requests to screen    */
                refresh();
                getch();

        /* part4 : wrapup                     */
                endwin();
}
:::::::::::::: hello3.c ::::::::::::::
#include       <stdio.h>
#include       <curses.h>

/*
 * hello3.c
 *      purpose using refresh and sleep for animated effects
 *      outline initialize, draw stuff, wrap up
 */
```

```
main()
{
        int     i;

        initscr();

        /*  do some screen operations */
                clear();
                for(i=0; i<LINES; i++ ){
                        move( i, i+i );
                        if ( i%2 == 1 ) standout();
                        addstr("Hello, world");
                        if ( i%2 == 1 ) standend();
                        refresh();
                        sleep(1);
                }


        /*  wrapup                      */
                endwin();
}
:::::::::::::: hello4.c ::::::::::::::

#include       <stdio.h>
#include       <curses.h>

/*
 * hello4.c
 *      purpose show how to use erase, time, and draw for animation
 */

main()
{
        int     i;

        initscr();

        /*  do some screen operations */
                clear();
                for(i=0; i<LINES; i++ ){
                        move( i, i+i );
                        if ( i%2 == 1 ) standout();
                        addstr("Hello, world");
                        if ( i%2 == 1 ) standend();
                        refresh();
                        sleep(1);
                        move(i,i+i);                    /* move back   */
                        addstr("             ");       /* erase line  */
                }


        /*  wrapup                      */
                endwin();
}
:::::::::::::: hello5.c ::::::::::::::
#include       <curses.h>

/*
 * hello5.c
 *  bounce a message back and forth across the screen
 */

#define LEFTEDGE       10
#define RIGHTEDGE      30
#define ROW            10
```

```
main()          /* hello5.c */
{
        char    msg[] = " Hello ";            /* notice padding spaces */
        int     dir = +1;
        int     pos = LEFTEDGE ;

        initscr();
        clear();

        while(1){
                move(ROW,pos);
                addstr( msg );                /* draw it */
                refresh();
                pos += dir;                   /* advance position    */
                if ( pos >= RIGHTEDGE )               /* check for bounce    */
                        dir = -1;
                if ( pos <= LEFTEDGE )
                        dir = +1;
                sleep(1);
        }
}
:::::::::::::: sleep1.c ::::::::::::::

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

/*
 * sleep1.c
 *              purpose show how sleep works
 *              usage   sleep1
 *              info    sets handler, sets alarm, pauses, then returns
 */

main()
{
        void    handler();

        printf("about to sleep for 4 seconds\n");
        signal(SIGALRM, handler);                     /* catch it    */
        alarm(4);                                     /* set clock   */
        pause();                                      /* do nothing  */
        printf("Morning so soon?\n");                 /* back to work       */
}

void
handler()
{
        printf("Alarm received from kernel\n");
}
:::::::::::::: hello6.c ::::::::::::::
#include       <curses.h>

/*
 * hello6.c
 *  bounce a message back and forth across the screen, uses millisleep()
 */

#define LEFTEDGE       10
#define RIGHTEDGE      30
#define ROW            10
#define TICKS          200                    /* 200/1000 sec */

main(int ac, char *av[])
{
        char    msg[] = " Hello ";            /* notice padding spaces */
        int     dir = +1;
        int     pos = LEFTEDGE ;
        int     delay = TICKS;

        if ( ac > 1 ) delay = atoi( av[1] );
        initscr();
        clear();
```

```
                                                /* (hello6. continued) */
        while(1){
                move(ROW,pos);
                addstr( msg );                  /* draw it */
                move(0,0);
                printw("pos = %02d", pos);
                refresh();
                pos += dir;                     /* advance position    */
                if ( pos >= RIGHTEDGE )                 /* check for bounce    */
                        dir = -1;
                if ( pos <= LEFTEDGE )
                        dir = +1;
                millisleep(delay);
        }
}

:::::::::::::: sigdemo.c ::::::::::::::
#include       <stdio.h>
#include       <signal.h>

/*
 * sigdemo.c
 *      purpose: show answers to signal question
 *      question1: does the handler stay in effect after a signal arrives?
 *      question2: what if a signalX arrives while handling signalX?
 *       question3: what if a signalX arrives while handling signalY?
 *       question4: what happens to getc() when a signal arrives?
 */


main(int ac, char *av[])
{
        void    inthandler(int);
        void    quithandler(int);
        char    input[100];

        signal( SIGINT,  inthandler );              /* set trap           */
        signal( SIGQUIT, quithandler );             /* set trap           */

        do {
                printf("\nType a message\n");
                if ( gets(input) == NULL )
                        perror("Saw EOF ");
                else
                        printf("You typed: %s\n", input);
        }
        while( strcmp( input , "quit" ) != 0 );
}
void
inthandler(int s)
{
        printf(" Received signal %d .. waiting\n", s );
        sleep(2);
        printf("  Leaving inthandler \n");
}

void
quithandler(int s)
{
        printf(" Received signal %d .. waiting\n", s );
        sleep(3);
        printf("  Leaving quithandler \n");
}
```

```
:::::::::::::: play_again5.c ::::::::::::::
#include       <stdio.h>
#include       <curses.h>
#include       <signal.h>

/*
 * play_again5.c
 *
 *      purpose: ask if user wants another transaction
 *       method: use curses to select crmode() and noecho()
 *               uses set_ticker() to send pulses
 *      returns: 0=>yes, 1=>no, 2=>timeout
 *         note: requires alarmlib.o -lcurses
 */
#define ASK            "Do you want another transaction"
#define TIMEOUT        5       /* in seconds  */

main()
{
        int     response;

        initscr();
        crmode();
        noecho();
        clear();
        response = get_response(ASK, TIMEOUT);       /* get some answer     */
        endwin();
        return response;
}

int     time_left = 0;

get_response( question , timeout )
char *question;
/*
 * purpose: ask a question and wait for a y/n answer or timeout
 *  method: set ticker to generate question marks
 * returns: 0=>yes, 1=>no
 */
{
        int     input;
        void    alarm_handler();

        time_left = timeout;                 /* set global var      */
        signal( SIGALRM, alarm_handler );    /* set handler         */
        set_ticker(1000);                    /* and ticker          */

        move(LINES/2, 10);
        addstr( question );
        addstr( " (y/n)? " );
        refresh();
        while ( time_left > 0 ){
                input = getch();                        /* use curseses        */
                if ( input == 'y' || input == 'Y' )   /* Y?          */
                        return 0;
                if ( input == 'n' || input == 'N' )   /* N?          */
                        return 1;
        }
        return 2;
}

void
alarm_handler()
{
        signal(SIGALRM, alarm_handler);                 /* reset trap  */
        addch('?');                                     /* prompt user */
        refresh();                                      /* and show it */
        time_left--;                                    /* adjust count        */
        if ( time_left == 0 )
        {
                endwin();
                exit(2);
        }
}
```

```
:::::::::::::: bounce1d.c ::::::::::::::
#include       <stdio.h>
#include       <curses.h>
#include       <signal.h>

/*
 * bounce1.c
 *      purpose animate using curses and alarm
 *      note    the handler does the animation
 *              the main program reads keyboard input
 *      compile cc bounce1.c alarmlib.c -lcurses -o bounce1
 */

/*
 * some global vars that main and the handler use
 */

#define MESSAGE " hello "

int     cur_row;        /* current row         */
int     cur_col;        /* current column      */
int     direction;      /* where we are going  */

main()
{
        int     delay;          /* bigger => slower    */
        int     ndelay;         /* new delay           */
        int     c;              /* user input          */
        void    on_ticker();    /* handler for timer   */

        initscr();
        crmode();
        noecho();
        clear();

        cur_row   = 10;               /* start here          */
        cur_col   = 0;
        direction = 1;          /* add 1 to row number */
        delay     = 1000;       /* 1000 ms = 1 second  */

        move(cur_row,cur_col); /* get into position   */
        addstr( MESSAGE );     /* note spaces         */
        signal(SIGALRM, on_ticker );
        set_ticker( delay );

        while(1)
        {
                ndelay = 0;
                c = getch();
                if ( c == 'Q' ) break;
                if ( c == ' ' ) direction = -direction;
                if ( c == 'f' && delay > 2 ) ndelay = delay/2;
                if ( c == 's' ) ndelay = delay * 2 ;
                if ( ndelay > 0 )
                        set_ticker( delay = ndelay );
        }
        endwin();
}

void
on_ticker()
{
        signal(SIGALRM, on_ticker);   /* reset, just in case */
        cur_col += direction;         /* move to new column  */
        move( cur_row, cur_col );     /* then set cursor     */
        addstr( MESSAGE );            /* redo message                */
        refresh();                    /* and show it         */

        /*
         * now handle borders
         */
        if ( direction == -1 && cur_col == 0 )
                direction = 1;
        else if ( direction == 1 && cur_col+strlen(MESSAGE) >= COLS )
                direction = -1;
}
```

```
:::::::::::::: bounce2d.c ::::::::::::::
#include       <curses.h>
#include       <signal.h>


/*
 *      bounce2d 1.0
 *
 *      bounce a character (default is *) around the screen
 *      defined by some parameters
 *
 *      user input:
 *                      s slow down x component, S: slow y component
 *                      f speed up x component,  F: speed y component
 *                      Q quit
 *
 *      blocks on read, but timer tick sets SIGALRM which are caught
 *      by ball_move
 */

#include       "bounce.h"

struct ppball the_ball ;

/**
 **     the main loop
 **/

main()
{
        int     c;

        set_up();

        while ( ( c = getchar()) != 'Q' ){
                if ( c == 'f' )             the_ball.x_ttm--;
                else if ( c == 's' ) the_ball.x_ttm++;
                else if ( c == 'F' ) the_ball.y_ttm--;
                else if ( c == 'S' ) the_ball.y_ttm++;
        }

        wrap_up();
}


set_up()
/*
 *      init structure and other stuff
 */
{
        void    ball_move();

        the_ball.y_pos = Y_INIT;
        the_ball.x_pos = X_INIT;
        the_ball.y_ttg = the_ball.y_ttm = Y_TTM ;
        the_ball.x_ttg = the_ball.x_ttm = X_TTM ;
        the_ball.y_dir = 1  ;
        the_ball.x_dir = 1  ;
        the_ball.symbol = DFL_SYMBOL ;

        initscr();
        noecho();
        crmode();

        signal( SIGINT , SIG_IGN   );
        put_a_char( the_ball.y_pos, the_ball.x_pos, the_ball.symbol  );
        refresh();

        signal( SIGALRM, ball_move );
        set_ticker( 1000 / TICKS_PER_SEC );   /* send millisecs per tick */
}
```

```
wrap_up()
{

        set_ticker( 0 );
        endwin();               /* put back to normal  */
}


void
ball_move()
{
        int     y_cur, x_cur, moved;

        signal( SIGALRM , SIG_IGN );          /* dont get caught now        */
        y_cur = the_ball.y_pos ;              /* old spot            */
        x_cur = the_ball.x_pos ;
        moved = 0 ;

        if ( the_ball.y_ttm > 0 && the_ball.y_ttg-- == 1 ){
                the_ball.y_pos += the_ball.y_dir ;    /* move */
                the_ball.y_ttg = the_ball.y_ttm  ;    /* reset*/
                moved = 1;
        }

        if ( the_ball.x_ttm > 0 && the_ball.x_ttg-- == 1 ){
                the_ball.x_pos += the_ball.x_dir ;    /* move */
                the_ball.x_ttg = the_ball.x_ttm  ;    /* reset*/
                moved = 1;
        }

        if ( moved ){
                put_a_char( y_cur, x_cur, BLANK );
                put_a_char( the_ball.y_pos, the_ball.x_pos,
                                the_ball.symbol  );
                bounce_or_lose( &the_ball );
                move(23,70);
                refresh();
        }
        signal( SIGALRM, ball_move);          /* for unreliable systems */

}

put_a_char( y, x , c )
char c;
{
        move(y,x);
        addch(c);
}

bounce_or_lose(bp)
struct ppball *bp;
{
        int     return_val = 0 ;

        if ( bp->y_pos == TOP_ROW )
                bp->y_dir = 1 , return_val = 1 ;
        else if ( bp->y_pos == BOT_ROW )
                bp->y_dir = -1 , return_val = 1;

        if ( bp->x_pos == LEFT_EDGE )
                bp->x_dir = 1 , return_val = 1 ;
        else if ( bp->x_pos == RIGHT_EDGE )
                bp->x_dir = -1 , return_val = 1;

        return return_val;
}
```

```
:::::::::::::: bounce.h ::::::::::::::
/**
 **     some parameters
 **/
#define BLANK          ' '
#define DFL_SYMBOL     '*'
#define TOP_ROW        5
#define BOT_ROW        20
#define LEFT_EDGE      10
#define RIGHT_EDGE     70
#define X_INIT         10             /* starting col               */
#define Y_INIT         10             /* starting row               */
#define TICKS_PER_SEC  50             /* affects speed      */

#define X_TTM          5
#define Y_TTM          8

/**
 **     the only object in town
 **/
struct ppball {
               int     y_pos, x_pos,
                       y_ttm, x_ttm,
                       y_ttg, x_ttg,
                       y_dir, x_dir;
               char    symbol ;

       } ;
:::::::::::::: alarmlib.c ::::::::::::::
#include       <sys/time.h>
#include       <signal.h>

/*
 *      alarmlib.c
 *                      timer functions for higher resolution clock
 *                      set_ticker( number_of_milliseconds )
 *                              arranges for the interval timer to issue
 *                              SIGALRM's at regular intervals
 *                      millisleep( number_of_milliseconds )
 *
 *      version 98.03.16
 */
extern int errno;

set_ticker( n_msecs )
/*
 *      arg in milliseconds, converted into micro seoncds
 */
{
       struct itimerval new_timeset, old_timeset;
       long    old;
       long    n_sec, n_usecs;

       n_sec = n_msecs / 1000 ;
       n_usecs = ( n_msecs % 1000 ) * 1000L ;

       new_timeset.it_interval.tv_sec  = n_sec;     /* set reload  */
       new_timeset.it_interval.tv_usec = n_usecs;   /* new ticker value */
       new_timeset.it_value.tv_sec     = n_sec  ;   /* store this  */
       new_timeset.it_value.tv_usec    = n_usecs ;  /* and this    */

       if ( setitimer( ITIMER_REAL, &new_timeset, &old_timeset ) != 0 ){
              printf("Error with timer..errno=%d\n", errno );
              exit(1);
       }
}
static void my_handler();

millisleep( n )
{
       signal( SIGALRM , my_handler);        /* set handler         */
       set_ticker( n );                      /* set alarm timer     */
       pause();                              /* wait for sigalrm    */
}
static void
my_handler()
{
       set_ticker( 0 );                      /* turns off ticker */
}
```