1. Anti-screensaver. This program is supposed to be used in the corporate environments where screensaver got introduced for machines with remote access. As a side effect of this, users are prompted for the passwords too many times: first, when the connection with the remote machine is being made, and later when the connection is already established.

Program is pretty simple and it'll be doing barely noticable movements with a mouse so that the screensaver can be tricked by "thinking" that system's user is still there. Simple invocation of a program should be enough.

2. First I'm including all sorts of standard header files which are found in pretty much every UNIX C program. I'm including `signal.h` and `unistd.h` since I'll need *signal()* and *getopt()* functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>      /* for getopt() */
```

3. Next I'll be using some of the X11 functions. X11 is a UNIX protocol for graphical displays, which every system is using. X11 comes to the picture when you're connecting to Xilinx login machine via VNC.

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xresource.h>
```

4. Before we start with general initialization, we have to understand that within UNIX every program can get an asynchronous signal, which is similar to hardware exception. One of the cases of getting a signal is pressing `CTRL-C`. We want to safely terminate the program when user pressed this combination, so we must declare the signal handler. Signal handler will be simple and will only memorize which signal it got:

```
static int g_got_sig = 0;
static void got_sig(int sig_num)
{
  g_got_sig = sig_num;
}
```

5. The body of the program follows.

```
int main(int argc, char **argv)
{
  ⟨ Local variables 6 ⟩;
  ⟨ Generic init 7 ⟩;
  ⟨ Command line arguments 8 ⟩;
  ⟨ Make checks 9 ⟩;
  ⟨ Open display 10 ⟩;
  ⟨ Main loop 11 ⟩;
  return 0;
}
```

6. Local variables. I'll be using *o* temporarily for command line processing with *getopt()*. *dpy* and *root_window* are handlers for X11 functions and its API. The *move* is just a temporary variable. I use it within an infinite loop and it'll be toggling between 2 values: 1 and −1. *move_const* is the value *move* will get multiplied by. Thus, $move_const * move$ will define the x and y offsets by which the mouse cursor will move.

⟨ Local variables 6 ⟩ ≡
```
int o;
Display * dpy;
Window root_window;
int move, move_const, sleep_amt;
```
This code is used in section 5.

7. Now we can proceed. Together with setting a signal handler, we also turn off the input/output buffering, which is problematic in case we need to debug the problem.

⟨ Generic init 7 ⟩ ≡
```
signal(SIGINT, got_sig);
setbuf(stdout, Λ);
setbuf(stderr, Λ);
```
This code is used in section 5.

8. Command line processing. We have a very simple command line arguments: `-c` and `-s`. The first one defines amount of pixel mouse will be getting moved back and forth. The second one modifies the second interval between mouse moves. In case user tried to supply invalid command, we present the error and finish. Upon successful completion of the parsing, we update the values of *argc* and *argv*.

```
⟨ Command line arguments 8 ⟩ ≡
  move_const = 1;
  sleep_amt = 5;
  while ((o = getopt(argc, argv, "c:s:")) ≠ −1) {
    switch (o) {
    case 'c': move_const = atoi(optarg);
      break;
    case 's': sleep_amt = atoi(optarg);
      break;
    default: fprintf(stderr, "unknown␣option␣'%c'\n", move_const);
      exit(1);
    }
  }
  argc −= optind;
  argv += optind;
```

This code is used in section 5.

9. Validity checks. To not fall into the user's trap, we much make sure user supplied correct amount of seconds for the wait interval. Should this value be too small, we fix it and inform the user about this. If the interval would be 0, our loop would try to execute too many X11 calls and the program could essentially kill our remote X11 session!

```
⟨ Make checks 9 ⟩ ≡
  if (sleep_amt ≤ 0) {
    printf("sleep_amt=%d, must be >= 1. Fixing to 1\n");
    sleep_amt = 1;
  }
```

This code is used in section 5.

10. We follow with simple call to X11 routines in order to open display.

```
⟨ Open display 10 ⟩ ≡
  dpy = XOpenDisplay(0);
  root_window = XRootWindow(dpy, 0);
  XSelectInput(dpy, root_window, KeyReleaseMask);
```

This code is used in section 5.

11. Main loop. We are finally ready to loop. We start from $move = 1$ and loop forever. Loop will be terminated when we get the signal from the user, in which case out *g_got_sig* handler will be called, and the *g_got_sig* variable will be set. Check at the beginning of a loop will make the loop to terminate.

```
⟨ Main loop 11 ⟩ ≡
  move = 1;
  for ( ; ; ) {
    if (g_got_sig ≠ 0) {
      printf("Got␣signal␣%d.␣Terminating\n", g_got_sig);
      break;
    }
    XWarpPointer(dpy, None, None, 0, 0, 0, 0, move * move_const, move * move_const);
    XFlush(dpy);
    if (move ≡ 1) {
      move = −1;
    }
    else {
      move = 1;
    }
    sleep(sleep_amt);
  }
```

This code is used in section 5.

12. Index.

⟨ Command line arguments 8 ⟩ Used in section 5.
⟨ Generic init 7 ⟩ Used in section 5.
⟨ Local variables 6 ⟩ Used in section 5.
⟨ Main loop 11 ⟩ Used in section 5.
⟨ Make checks 9 ⟩ Used in section 5.
⟨ Open display 10 ⟩ Used in section 5.

UNSAVER