1 Objectives

The objectives of this lab are to:

1. have more practice with the C language;
2. declare, implement and call your own functions;
3. learn how to use the random number generator;
4. learn about ANSI escape codes (which we have not discussed in class).

2 Program Requirements

Big picture: write a program that displays a requested number of '*' at random locations of the terminal.

2.1 Interface Requirements

1. The program shall be called `dots`.
2. The command-line syntax shall be:

   ```
   dots number
   ```

   where:

   `number` = the number of random dots to display
3. The requested number of dots can be any number between 1 and 1000 (inclusive). Any other value is an error.
4. The terminal shall be cleared by your program before any dots are displayed.
5. The top-left corner of the screen is the coordinate (1,1); the next character to its right is at the coordinate (2,1), and so on. In other words, the terminal has a coordinate system as shown below:

(1,1)	(2,1)	(3,1)	(4,1)	…	(80,1)
(1,2)					
(1,3)					
…					
(1,24)					

6. When displaying each dot, the location shall be generated randomly. This coordinate must fall within the allowable area of the screen.
7. There is the possibility that two randomly generated dots will occupy the same coordinate.
8. Wait one second between the drawing of each dot. Hint: see "`man 3 sleep`".
9. After all the dots have been displayed, the cursor shall be moved to the first character in the bottom row of the screen before the program exits.
10. Any errors encountered, such as an invalid input from the user, should result in 1) a meaningful error message displayed to the screen, and 2) exiting the program with an error code of 1. Exit at the first detected error.

2.2 Internal Requirements and Notes

1. The program shall be implemented in a file called "dots.c".
2. In your code, determine the size of your terminal (the max characters across and down) by including the following code **in the appropriate places**.

```
#include <sys/ioctl.h>



        int max_rows = 0;
        int max_cols = 0;
        .
        .
        struct winsize win;
        ioctl(0, TIOCGWINSZ, &win);
        max_rows = win.ws_row;
        max_cols = win.ws_col;
```

3. No global variables are allowed.
4. The coordinates for the starting point of each dot shall be generated using the `random` library function. As stated above, these coordinates must fall within the allowable portion of the screen.
5. The random number generator will be seeded using the `srandom` library function so that the pattern of dots will be different for each execution of the program. This is done with the following code (that is only called once): `srandom(time(NULL));`
6. To display the dots, you shall declare and use a function called `display_dots` with the following prototype:

```
void display_dots(int num_dots);
```

7. To immediately see each dot after it has been printed, use the following code after the dot has been printed to the screen: `fflush(stdout);` If you don't put in this code, then you may not see the dots until after the program terminates.
8. If you need to stop your program before it finishes its execution, try pressing Control-C (sometimes referred to as Ctrl-C or ^C).

2.3 Screen Manipulation Details

Terminals in Unix (and somewhat in the Windows command-line) generally support what are called *ANSI escape codes*. These are character sequences that are interpreted by the terminal to provide functionality, such as cursor movement and colored text. Escape sequences start with the "`\033[`" string, which is then followed by other characters to generate the requested action(s). You will need to use the following escape codes:

1. To clear the screen, do the following in the appropriate places:

```
#define CLEAR_SCREEN "\033[2J"
        ...
        printf(CLEAR_SCREEN);
```

2. To move the cursor to row y, column x, use `printf` to print the following: "`\033[y;xH`" (where 'y' and 'x' are replaced with the desired location). For example:

```
#define MOVE_CURSOR "\033[%d;%dH"
        ...
        printf(MOVE_CURSOR, y, x);
```

For more escape codes, see https://en.wikipedia.org/wiki/ANSI_escape_code.

2.4 Makefile Requirements

You will write a Makefile that will be able to do three things: 1) create the `dots` program, 2) delete the `dots` program, and 3) create `proj2.tar` (the file you need to turn in). Specifically, the Makefile shall have the following targets:

1. All
   When "make" is entered on the command line, it only tries to create the `dots` program.
2. dots
   When "make dots" or "make" is entered on the command-line, this target will create the `dots` program.
3. clean
   When selected, this target will delete the `dots` program (**NOT**, dots.c)
4. `dist`
   When selected, this target shall create `proj2.tar`, which will contain "Makefile" and "dots.c".

2.5 Optional Fun

1. Color
   Figure out how to change the color of each dot via the escape codes.
2. Note that this may be optional this time, but may be required in a future project.

3 Submission

Before the deadline, submit to Sakai `proj2.tar`, (which will include the following):

1. dots.c
2. Makefile

4 Grading

Grading will be based on the following **guidelines**:

1. The code compiles without errors (20).
2. The code compiles without warnings when the `-Wall` compiler option is used (10).
3. No segmentation faults or other crashes occur when the code is run (10).
4. The code handles valid input as specified in Section 2. In particular, the following tests of valid input will be performed on a terminal that has been resized from the default size (20):

```
a. ./dots 1
b. ./dots 10
c. ./dots 30
d. ./dots 1000  # I will turn off the sleep function
```

5. The code handles **in**valid input as specified in Section 2. In particular, the following tests of invalid input will be performed (20):

```
a. ./dots
b. ./dots hello
c. ./dots hello world
d. ./dots 3rd
e. ./dots 3g
f. ./dots 0
g. ./dots -1
h. ./dots 1001
i. ./dots 20000
```

6. A working Makefile was provided, as specified in Section 2. (10)
7. The `strtol` library function was used to **verify valid input** and convert the input string to a number. (5)
8. Code complies with the style guide. (5)
9. Deductions may occur in the following situations:
    a. The code appears to be hard-coded to give the correct answers to the test input.
    b. Although the proper output appears, the approach you used does not match the way it was specified.
    c. I reserve the right to include other reasons I have not yet encountered.