1 Objectives

In addition to a general objective to practice the C language, the objectives of this lab are to learn how to:

1. declare, implement and call your own functions.
2. dynamically allocate memory.
3. use ANSI escape codes (which we have not discussed in class).

2 Program Requirements

Big picture: write a program that displays a requested number of horizontal lines at random starting locations of the terminal.

2.1 Interface Requirements

1. The program shall be called `lines`.
2. The command-line syntax shall be:
   ```
   lines number length
   ```
   where:
   `number` = the number of lines to display
   `length` = the length of each horizontal line
3. The number of lines can be any number between 1 and 1000 (inclusive).
4. The length must be a number between 1 and 40 (inclusive).
5. Each line shall be created using the dash for each unit of length.
   For example, a line of length 4 would be: `----`
6. **The bottom line of the screen is out of bounds for displaying a line.**
7. The terminal shall be cleared by your program before any lines are displayed.
8. 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.
9. When displaying each horizontal line, the location of the left-most edge of the line shall be generated randomly. This coordinate must fall within the allowable area of the screen.
10. Because the coordinates are generated randomly, the desired line may not fit entirely on the screen. For example, if a line of length 20 has been requested, but the coordinate is only three characters away from the right-most edge of the terminal, then the line shall stop at the right edge of terminal (i.e., it shall not wrap around to the next line).
11. Subsequent lines may overlap previously displayed lines.
12. Wait one second between the drawing of each line. Hint: see "`man 3 sleep`".
13. After all the lines have been displayed, the cursor shall be moved to the first character in the bottom row of the screen before the program exits.
14. 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. Exit at the first detected error.

2.2 Internal Requirements and Notes

1. The program shall be implemented in a file called "lines.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>
.
.
// Global variables
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. The only global variables allowed in your program are those shown above: `Max_rows` and `Max_cols`.
4. The coordinates for the starting point of each line 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 lines will be different for each execution of the program. This is done with the following code: `srandom(time(NULL));` The coordinates for the start of all lines shall be determined **before** any lines are displayed. These random coordinates shall be saved in dynamically allocated memory (i.e., you are **required** to use `malloc`).
6. When displaying the lines, you shall declare and use a function called `display_line` that can display **one** line at a time, taking the following inputs: col, row, length (where 'col' and 'row' are the coordinates for the left-most part of the line to display).
7. To immediately see each line after it has been printed, use the following code after the line has been printed to the screen: `fflush(stdout);`
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, use `printf` to print the following: "`\033[2J`".
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:

```
printf("\033[%d;%dH", 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 `lines` program, 2) delete the `lines` program, and 3) create dist2.tar (the file you need to turn in). Specifically, the Makefile shall have the following targets:

1. all
2. lines
   When selected, or when "make" is entered on the command-line, this target will create the `lines` program.
3. clean
   When selected, this target will delete the `lines` program.
4. `dist`
   When selected, this target shall create `dist2.tar`, which will contain "Makefile" and "lines.c".

2.5 Optional Fun

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

3 Submission

Submit dist2.tar to Sakai, (which will include the following):

1. lines.c
2. Makefile

4 Grading

Severe deductions shall be given in the following situations:

1. Compilation errors (i.e., your code will not compile).
2. No Makefile was provided.
3. The program crashes when bad input is given.
4. You wrote the `display` function such that it was only called once.
5. You did not use `malloc` to allocate memory for the line coordinates.
6. Improper handling of dynamic memory (such as, you did not use `free` to de-allocate your memory after you were done with it).
7. I reserve the right to include other reasons I have not yet encountered.

Other deductions of various degrees shall be given in the following situations:

1. Warning statements are seen when your code is compiled.
2. Numerous and obvious style violations. See the style guide.
3. The Makefile does not work as specified.

4. A displayed line wraps around to the next line.
5. A displayed line uses the bottom row of the screen.
6. The terminal is not cleared before the lines are displayed.
7. Given the same inputs, your program produces the same picture instead of a random placement of lines.
8. The following **invalid** input does not produce an error message or does not return value of -1:
    a. ./lines invalid number of arguments
    b. ./lines 12 invalid number of arguments
    c. ./lines 12 4 8 invalid number of arguments
    d. ./lines 12 41 invalid length
    e. ./lines 2000 2 invalid number of lines
    f. ./lines two 4 invalid number of lines
    g. ./lines 12 abc invalid length
9. The following **valid** input does not produce the correct results and a return value of 0:
    a. ./lines 1 1
    b. ./lines 1 2
    c. ./lines 2 20
    d. ./lines 30 20
10. I reserve the right to include other reasons I have not yet encountered.