Collaboration

You must implement this quiz with a partner. You should have listed a partner on the Google form available on Moodle *at class time*, and received a confirmation message regarding your partner from the professor.

Deliverables

You should deliver a **PDF** file addressing items 1 and 2, and a **compressed file** containing all the files originally handed in the skeleton code, completed according to the specifications below.

1. [2 pts] Please indicate appropriately:

   ☐ I looked at the syllabus and I am familiar with the grading procedures, textbook requirements, and tentative course schedule.

2. [3 pts] Textbook, Chapter 2 exercise 5.[1]

3. [3 pts] Textbook, Chapter 2 exercise 6.[2]

4. [12 pts] On Intel x86 machines, when a kernel boots, it prints information by writing to a video memory buffer present at address 0xB8000 in memory. This location comes from the PC standard defined by IBM. The video memory buffer contains 80x25 characters, accounting for 25 lines, each 80 characters long.

   In the skeleton code, you have a *video memory emulator* implemented using a library called `ncurses`[3]. If you look at `terminal.c`, you will see a variable called `terminal_buffer`, that has 80x25 entries. The first 80 entries of that buffer contain the characters in line 0, the next 80 entries contain the characters in the line 1, and so on. In general, the character at row r (0-based indexed), column c (0-based indexed) will be at position $r * 80 + c$.

   Note that each entry in the buffer is of type `uint16_t`. In C, the standard types (`char`, `int`, `long int`) do not have a fixed bit width across different architectures, but each entry of the video memory buffer should be exactly a 16-bit integer. Luckily, the C99

[1] Answer in the book if you do the assigned readings on Moodle.
[2] Answer in the book if you do the assigned readings on Moodle.
[3] If you are interested in the details of `ncurses`, take a look at `http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/`

standard has a header file containing types guaranteed to be of a certain width in bits: the `stdint.h` header file. Take a look at the available types here:

`http://en.cppreference.com/w/c/types/integer`

Each entry of the video memory buffer is called a *buffer entry*. It is a 16-bit integer where:

(a) The lower 8 bits (bits $0 \ldots 7$) of each buffer entry contain the ASCII code of the character that should be present in the associated row and column.

(b) The higher 8 bits of each buffer entry contain a *color combination* of foreground and background specifications.

   i. The lower 4 bits of the color combination contain the foreground color. The colors are described in `terminal.h`.

   ii. The higher 4 bits of the color combination contain the background color. The colors are described in `terminal.h`.

   Note that the foreground and background vary from 0 to 7, which is exactly what we can represent in 4 bits.

In the top of `terminal.c` you see variables `position_row` and `position_col`, indicating the *current* cursor position in your terminal. Note how the `terminal_clear()` function initializes these positions to $(0, 0)$, which is the **top-left** of the screen. You also see a variable called `color` in the top of the file. That variable contains the 8 bits of the *current* color combination, but we store it in a 16-bit integer for convenience: with a 16-bit integer, we can shift the valid, lower 8 bits of `color` to the upper 8 positions in order to form each buffer entry, as described above. See how that procedure is done in `terminal_clear()` as well. Anything printed with the `terminal_write()` function is printed at the *current* position and using the *current* color combination.

Your task. You have to fill two functions.

(a) Fill the `terminal_setcolor()` function to initialize the color combination according to the parameters passed to the function. This is a one-line code that uses bit-shift operations in C.

(b) Fill the `terminal_write()` function. You should fill

```
terminal_buffer[BUFFER_POSITION(position_col, position_row)]
```

with the appropriate buffer entry as defined above (lower 8 bits for the character's ASCII code, higher 8 bits for the current color combination). You should also change `position_col` (incrementing by one) and `position_row` according to the specification given below. Note that `position_row` can vary between 0 and `VGA_HEIGHT - 1` and `position_col` can vary between 0 and `VGA_WIDTH - 1`.

i. If the new character makes you go past the end of a line, move to the next line incrementing `position_row` and setting `position_col` to 0.

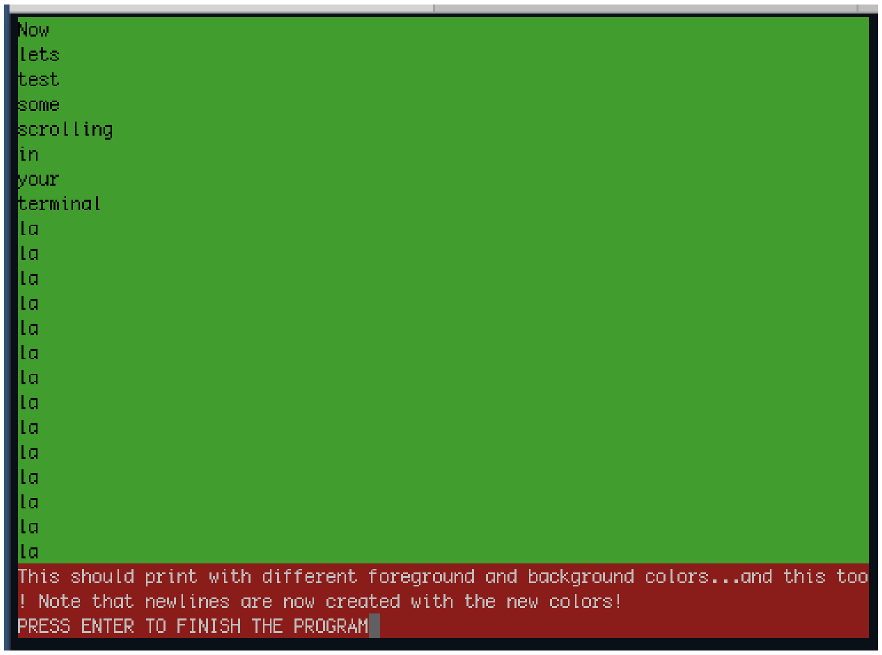
ii. If the character is a newline character (`\n`), move the current cursor position to the next line as above.

iii. If the character is tab character (`\t`), increment `position_col` to the next multiple of 8, and move the current cursor position to the next line as above, if necessary.

iv. If you are **past the last line**, scroll! Move all buffer entries indexed by lines 1...`VGA_HEIGHT - 1` to 0...`VGA_HEIGHT - 2`, and then clear the last line. To clear the last line, fill only the last line positions in `terminal_buffer[]` with blanks, just as done in `terminal_clear()`.

v. In the end of your `terminal_write()` function, call `move_cursor()` to move the cursor to the updated current position, and call the `draw()` method that will fill up the screen with the appropriate contents.

Here's a screen shot of how your screen should look like after all the messages from `main()` are printed. Before each message is printed, you have to press a key. Look at the code in the `main()` function.



```
Now
lets
test
some
scrolling
in
your
terminal
la
la
la
la
la
la
la
la
la
la
la
la
la
la
This should print with different foreground and background colors...and this too
! Note that newlines are now created with the new colors!
PRESS ENTER TO FINISH THE PROGRAM
```

Compiling and Running

The skeleton file provided for this quiz contains a Makefile, which consists of a set of rules to direct GCC to compile your program. The productive way to code is to use something like **VSCode.**[4] To use VSCode to compile/run your program, install the following extensions:

(a) C/C++

(b) Clang-format (if you are in macOS, clang is a better compiler than GCC)

(c) Code Runner

(d) GitLens (use version control!)

(e) Partial Diff (if you spend 5min to learn it, you'll see how nice this is)

Here's an intro to the tool:

(a) `https://code.visualstudio.com/docs/introvideos/basics`

(b) `https://code.visualstudio.com/docs/languages/cpp`

[4]Atom is good too.