CS 105 (C++)

Assignment 3: ASCII Shading

I. Overview

In this assignment, you will write a program to perform a kind of *image to ASCII conversion*, a form of ASCII art.

Given a stream of image shading values and dimensions, the code you write will produce images shaded with ASCII text, like the one above. The following are some important details of what your program must do.

- Your program's input will be an unbroken stream of numeral characters (from `'0'` to `'8'`) acquired using `getchar`.
- Your program will take two arguments: width and height of the output image (in that order).
- Your program's output will be properly arranged (according to the given dimensions) lines of ASCII shading characters from the following palette:

input char	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'

output char	' '	'.'	':'	'c'	'o'	'C'	'O'	'8'	'@'

- If any input characters other than those listed here are encountered, your program should print an error message, then quit.

In order for you to gain experience in some of the required topics for this assignment, your program's implementation must include the following features:

- Your program must be compiled from 3 source files:
  - `main.c` (Handles input and output, as well as top-level program logic.)
  - `shade.h` (Declares the function `shade`, which converts from input char to output char as shown in the table above.)
  - `shade.c` (Defines the function `shade`, as declared in `shade.h`.)
- Your `shade.h` file must contain the proper preprocessor directives to prevent multiple inclusion. (See Subsection 4.11.3 of *The C Programming Language*.)
- The function `shade` must be declared exactly as follows.
  - return value: `int`, with `0` indicating failure (due to invalid input char), and nonzero indicating successful conversion.
  - first argument: `char`, containing the input character.
  - second argument: `char *`, containing a pointer to a variable in which the output character can be stored. (This is a common method for cases like this, in which a function needs to "return" more than one value at a time.)
- Your program must read image width and height from its arguments. (See Section III, below, for details.)
- Your program must display a usage message if the wrong number of arguments is used.

Note: Your program's input must match the background shade (dark or light) that the output will be viewed on. To this end, I have provided two versions of the test input file (one for each background shade). If your results will be viewed on a light background, use `for_light_bg.txt` as your input. For results to be viewed on a dark background, use `for_dark_bg.txt` as your input. Both of these input files should be used with values of 95 and 75 for the width and height respectively. Note also that your results must be viewed in a constant-width font.

So, for example, assuming that your program is named `a3`, if you plan to view your results on a dark background (recommended), your code should be run on the given input file as follows.

```
cat for_dark_bg.txt | a3 95 75
```

Instead of simply viewing the results directly, you can use the following command to redirect your program's standard output to a new file called out.txt.

```
cat for_dark_bg.txt | a3 95 75 > out.txt
```

(Note that a simple way to view light-background results as if they were on a dark background is to highlight them.)

II. Grading

- Minimum Requirements
  - Correctly translate input numeral characters to ASCII shading characters, with proper formatting.
  - Proper declaration and definition of function `shade`, as described above.
  - Your work must be submitted in files with the following names:

    `main.c`

    `shade.h`

    `shade.c`
  - These files must compile on a department UNIX machine with the following command (as described in Section III, below):

    `cc main.c shade.c -o a3`
  - Before evaluation, your code must be submitted via `turnin`, using the following command on a department UNIX machine:

    `turnin --submit dlessin a3 main.c shade.h shade.c`
- Graded Elements
  - Proper use of command line arguments in `argv` to acquire dimensions.
  - Proper use of `argc` to display usage when appropriate.
  - `shade.h` must use preprocessor directives to prevent multiple inclusion.
  - Proper use of pointers/addresses to "return" values from function `shade`.
  - Quit with error message if an improper input character is encountered.

III. The More You Know

The following are some additional items that may be very important for you to know about this assignment.

- Your program will receive its arguments as character strings stored in `argv`, but how can you convert from a character string like `"95"` to the integer value 95? After including `stdlib.h`, you can call the function `atoi` to perform the conversion for you. The required code might look something like this:

```
#include <stdlib.h>
...
width = atoi(argv[1]);
```

- This assignment is the first time you'll have to combine multiple source files into a single executable. The most straightforward way to do this is to simply list all the source files (except for .h files, which are included using preprocessor directives) as arguments to `cc`. So, for example, to compile `main.c` and `shade.c` into a single executable called `a3`, you can use the following command:

```
cc main.c shade.c -o a3
```