Den Thermostat Narrative

The den thermostat program functions as an MVC program, where the model is the fakenews simulated sensor provided to us, the controller is the den.c program I wrote, and the view is the output that is displayed in the terminal while the program is running via printf calls. Any time we request a temperature reading, we are calling the fakenews() function, and any time we output information to the view, we are calling printf() to print info to the user.

The program starts by initializing some global variables and some functions for code clarity and reusability. The logic of the program is as follows:

Display “Welcome to den” and user instructions on startup (printed to the view).
Initially, we set the **set_temp** variable to be the default value of 72, and display that **set_temp** to the view. We then get an initial reading from the model, and assign that value to **current_temp**. This initial **current_temp** is displayed to the view. We also initialize a char variable with enough space to store any expected input from the user, called ‘**input**’, and set input to be non-blocking so that we can receive user input without causing the program to stop and wait for it.
At this point in the program, we enter the while(1) loop, where we remain until the program is exited.
Within the while loop, we start by getting a new value from the model, assigning this value to a variable called **new_temp**. We check whether **new_temp** == **current_temp**. If not, then we assign the value of **new_temp** to **current_temp**, and display this new **current_temp** to the view. Otherwise, if **new_temp** is the same value as **current_temp**, we continue.
Then, we check whether we have received any user input, by reading from STDIN and checking whether anything was read. If not, we continue, taking us back to the start of the loop where we get a new temperature reading from the model. If there *is* user input, then we process that input and act accordingly. There are three options within the logic of processing this input:

1. If **input**[0] == ‘x’, the user has entered ‘x’ to exit the program. We call exit(0) and the program ends.
2. If **input**[0] == ’t’, and **input**[1] == ‘ ‘, then the user has entered the code to set a new temperature. We convert the string stored in **input**, starting at the index **input**[2], to an integer, and assign that integer value to **set_temp**. Now that we have a new **set_temp**, we display that new set temperature to the view, and also display the **current_temp** to the view. Then we continue, returning to the start of the loop where we get a new temperature reading from the model.
3. If the value stored in **input** does not meet either of the above criteria, then we print an “invalid input” message to the view, and continue, returning to the start of the loop where we get a new temperature reading from the model.

References:

To complete this assignment, I relied on google/ChatGPT to provide me with the syntax to accomplish certain things:

- ChatGPT/google AI provided me with the syntax for converting a string to an integer value using the atoi() function.
- ChatGPT/google AI provided me with the syntax and more generally the method for getting non-blocking keyboard input using **fcntl.h**. I was having trouble getting this program to function correctly, because when I tried to use **scan** or **fgets**, it would stop the program and wait until there was user input. The non-blocking method provided by ChatGPT allowed me to check for user input without blocking/stopping the program and waiting until there was input.
- ChatGPT/google AI provided me with the syntax I used to add to the makefile, such that running the **make** command would compile the executable required to run the program.