Purpose: More practice with arrays & pointers. Introduction to "`structs`" and serial I/O.

Assignment: For this lab your program will get a key from the keypad and then play one of two songs depending on if a '1' or '2' key is hit. It will then loop back to get another key. For the "A" and "B' grade you will investigate using a serial device which will listen for characters (typed on the PC), echo them (i.e., send them back to the PC), and then play one of two songs ("B" grade) or individual notes ("A" grade) depending on which key is received.

Prelab: CodeLab problems related to this lab must be completed before coming to lab.

Notes:
The web page gives partial code including several `struct` and function declarations that you will use in your program. Use as given.

There are two `structs` shown: a note and a song. The `note` gives the information for a note, specifically the note number (index into a frequency table), and duration in 1/8ths of a second. A song gives the information for a song, specifically the number of notes, and an array of the actual notes. Note we placed an artificial limit of 30 notes per song.

Two songs and a frequency table are also given. The frequency table gives the frequency in cycles per second for each note number. (E.g., note number 0 has a frequency of 262 Hz.)

The speaker will be hooked up to GPIOA pin 10. You will need to make this pin an output. (As a hint, see how we made the bar-led bits outputs.) To make sound you will toggle this pin up and down. E.g., to play a tone of 440Hz, toggle this pin 880 times per second (440 complete cycles). Create a "`#define TOGGLE_SPKR`" at the top of your code that will toggle this bit.

Your main program will, in an infinite loop, wait for a key to be entered on the keypad and then if the key was '1' then play the Stein Song, and if it is '2', play "Mary Had a Little Lamb". Then loop back and do it again.

In addition to functions previously written you will need:

`playnote()` will play one note: First compute the number of times to toggle the speaker (full cycles – low then high). This is given by the frequency times the note duration (divide by 8 as duration is given in 1/8ths of a second.) Then compute the time between toggles. This is half of 1,000,000 divided by the frequency. Do math as best possible to avoid round-off errors. Then toggle the speaker the computed number of times. For each toggle, drive the speaker low (or toggle it), delay the computed time, drive the speaker high (or toggle it), delay the computed time. Delay 10ms at the end to provide a slight break between notes.

`playsong()` will simply play the notes of a song (passed to it as an argument) using a loop.

Doing the above will get you the “C” grade. For the “A” and “B” grades, first get the “C” grade checked off. Additional code will receive characters from the keyboard, echo them back to the PC, then play a song (“B” grade) or note (“A” grade). For the “B” grade, the '1' and '2' ASCII characters typed in the terminal window will select a song as before. For the “A” grade, the programs starts using the keypad, as in the “C” grade code, but pressing the ‘A’ button on the keypad switch will switch to “piano” mode. In this mode each digit pressed in the terminal window will play a corresponding note for 1/8 second. (`q’, `w’, `e’, `r’, `t’, ‘y’, ‘u’, ‘i’, ‘o’ correspond to notes C, D, E, F, G, A, B, C, D, respectively, and ‘2’, `3’, ‘5’, ‘6’, ‘7’, ‘9’ correspond to the notes C#, D#, F#, G#, A#, C#, respectively). `playnote()` will suppress its 10ms delay while in this mode. Entering ‘z’ will terminate this mode, returning to keypad control of which song is to be played.

For the A/B grades, you will need the following additional functions:

A/B grade details:

`serialRead(void)` will receive a character from the serial port. It works as follows: wait for the `RXNE` bit (bit 5) of `USART2->SR` to go to 1, then read the character in `USART2->DR` and return it.

`serialWrite(unsigned char c)` will transmit a character out the serial port. It works as follows: wait for the `TXE` bit (bit 7) of `USART2->SR` to go to 1, then write the character to the `USART2->DR register`.

NOTE: We will use USART2 which is enabled by default on our boards, but we will need to include the “low-level” code. For reference on the registers we are using see the documentation in Section 19.6 beginning page 548 of the reference manual (also linked on the webpage) https://www.st.com/resource/en/reference_manual/dm00096844-stm32f401xb-c-and-stm32f401xd-e-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

As in Lab 10, after you create your project from the `.ioc` file, click on the `<projectname>.ioc` item in the project explorer to open it up. Set up Timer 2 as we did for Lab 10. As part of this, we included the low level code for TIM2. For this lab the USART is already set up, but the low level code is not included by default. So, while you are in the “Project Manager”, “Advanced Settings”, as you did for TIM2 also expand USART and click on “HAL” and change it to “LL”. This should be all you need to do .

When your board is running, you can communicate through the serial port, using an application on your computer. For Linux, I recommend “screen” and for PC’s I recommend “PuTTY”. I’m not sure about Mac’s but I know it is there. Further details can be given in lab.