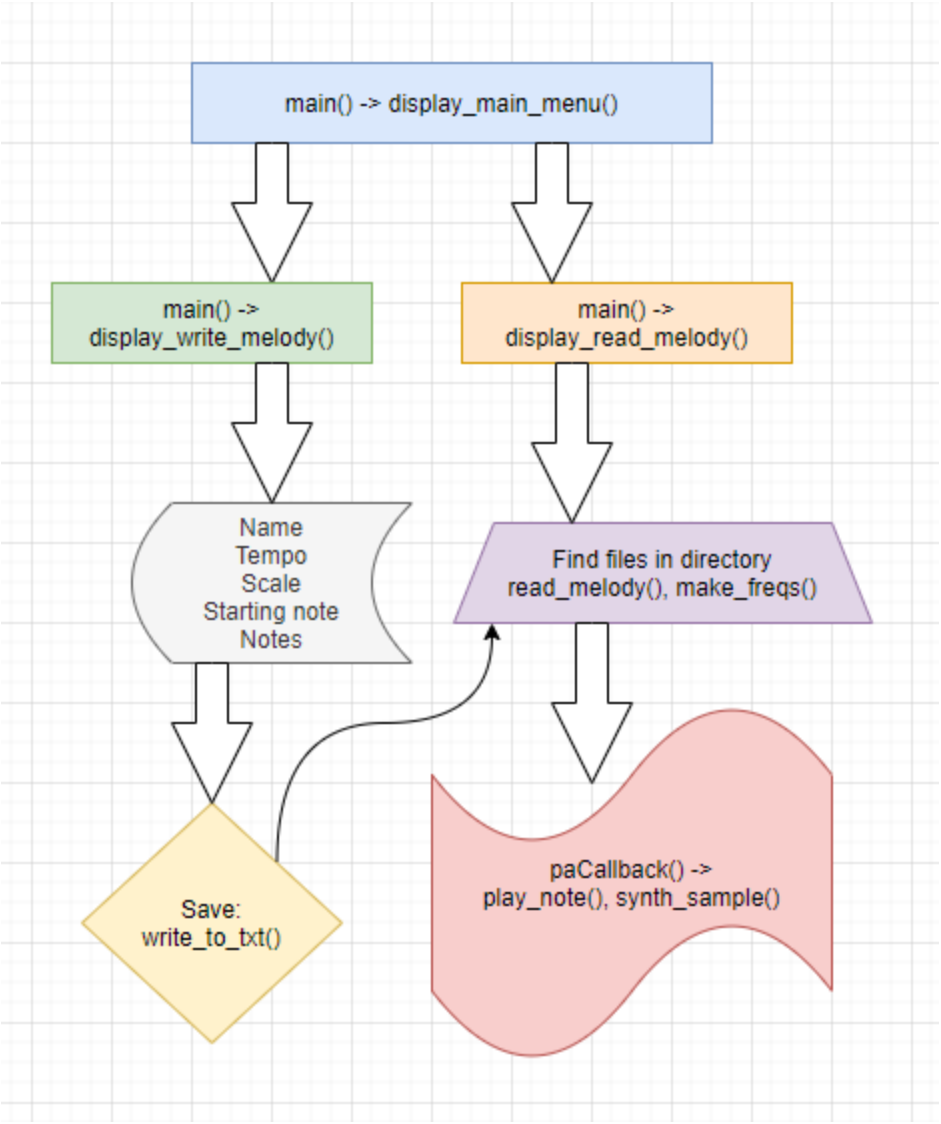
Pentaseq - the pentatonic melody composer/performer
by Jeff Holland

The objective of this program is to give the user an interface to write, save, and play back melodies in the pentatonic scale (major or minor), restricted to one octave, with 16 notes. These melodies are stored as .txt files. In the melody writing interface, the user can specify all of the following parameters:

- **Name** - this will be the name of the melody file, with ".txt" appended to it.
- **Tempo** - this is specified in beats per minute (BPM) and determines how fast the notes will be played. Because the notes are sixteenth notes, slower tempos are preferable. For example, 60 BPM corresponds to 1 quarter note or 4 sixteenth notes per second.
- **Scale** - this can be specified to either 1 for major pentatonic, or 2 for minor pentatonic.
- **Starting note** - this is specified as a MIDI number; for example, 48 corresponds to C3.
- **Notes** - the user can enter the notes in a grid with 6 rows and 16 columns. The rows correspond to 6 notes in the pentatonic scale (from the starting note to an octave above it). If no note is selected in a particular column, it is treated as a rest and the previous note is allowed to ring out.

Songwriters could use this program to try out melody ideas. Many pop and folk melodies are in the pentatonic scale, so it could be useful for songwriters to write melodies restricted to this scale. However, future versions of this program could include different kinds of scales and more octaves to offer more versatility.

The diagram on the next page shows the basic functionality of this program, with all of the major functions that I wrote myself. Without going into too much detail, I will also provide a verbal description of the program's flow.



The **main()** routine instantiates all of the major structs: most notably the Synth and Melody structs, as well as the Buf struct which will be passed into the Portaudio callback, and three instances of the Window struct which is native to Ncurses. These three Ncurses "windows" are the main menu, the *write melody* window, and the *play melody* window.

After all of the necessary parameter initializations, the **display_main_menu()** function allows the user to select an option from the main menu: either "Write melody", "Play melody", or "Exit." This function called in a while loop which continues until the user selects an option by pressing enter.

The *Write melody* menu prompts the user to enter a name, tempo, scale, and starting note for the melody. Once they have entered all of these parameters, the screen clears and a grid made of the character "x" appears on screen. At this point the main() routine has entered into another while

loop which calls **display_write_melody()**, which displays the grid and allows the user to navigate around the grid using the arrow keys and select notes to be written to the melody.

The user can then press F1 to save the melody, r to go to the "Play melody" menu, or q to quit the program. When the user presses F1 to save the melody, the function **write_to_txt()** is called, which saves all of the melody data to a txt file in a format that can be read by the program, with a filename specified by the user.

The *Play melody* menu gives the user a list of all .txt files in the current working directory, if there are any. This is accomplished using the **opendir()** and **readdir()** functions from the "dirent.h" library. After these functions find the files and write their names to an array, the **display_read_melody()** function is called to either display the list of found .txt files for the user to choose from, or if no .txt files were found in the directory, the program simply says "No melodies found" and prompts the user to quit the program.

If melodies were found, the user can select a file from the list and press enter to play it. When they do so, the **read_melody()** function is called to read the melody data from the .txt file, and then the **make_freqs()** function is called to convert the melody notes into frequencies. Once that happens, the **paCallback()** function is able to read the melody and begins making sound. It plays the melody on loop. At the last minute, I added a recording feature - the user can simply press the 'r' key to start recording, and quit the program to stop. The user can press q at any time to quit the program.

In terms of runtime complexity, I would estimate that this program runs at linear complexity, or O(n). The size of every melody is fixed - there is no way to have a melody with more than 16 notes. So the runtime of the program depends only on how many melodies are put into it. The program's operations are not complex: it takes in keyboard input from the user, reads and writes .txt files, assigns data to structures, synthesizes audio data, and sends the data as audio output. All for loops are based on constant values like the number of frames in a buffer (1024) or the number of notes in a melody (16), and while loops are simply used to display a menu while a user inputs keystrokes.

This program was implemented using Cygwin on Windows 10. It can be compiled using the build script included in the source code folder: **./build.sh**. It can then be run with **./pentaseq**. It does not take any command line arguments. The required libraries are stdio.h, stdlib.h, string.h, math.h, dirent.h, portaudio.h, ncurses.h, and sndfile.h.