The assignment is to be able implement functions that will ultimately read in a song string and play the song string. My code handles the defined sequence of events properly and is capable of sounding correct notes. The interface is shown in RealTerm via UART serial connection. The notes are represented by values A through G and R will represent a rest in the song. The durations of the notes or rests will be between 0 seconds and 31 seconds. My MatchSong() function does not always work. It is definitely capable of handling one-token searches. Please build by code twice to remove warnings, or alternatively build in each file individually in this order: U0_UART.h, U0_UART.c, music.h, music.c, main.c .

A few obstacles I had to overcome was handling memory overflow in the data memory. I resolved this issue by first implementing printf_P(PSTR()) functions to use program memory for print statements. I then shortened the maximum title character length from 128 to 32, which cleared more than half of my data memory usage. Another issue I had was that I was accidentally passing song[] (1D array) as an argument instead of a 2D array, song[NUMBER_OF_SONGS][MAX_SONG_LENGTH]. After adjusting this, my menu printing functionality was fixed. I had issues coding the MatchScore due to strange occurrences when comparing different tokens. The tokens became broken and illegible after having more than one token. If I had more time, I would probably try to wipe the strings used to store the tokens prior to comparing the query and database strings.

The main functionality of the code is creating songs, playing songs and displaying the stored songs. The main menu that holds these three choices for the user is inside an infinite while loop. The flow the user will follow is described below:

```
CreateSong() {
     RequestInputForNewTitle;
     ReplaceTitle;
     RequestInputForNewSongString;
     StoreNewSong;
     return;
}

PlaySong() {
     GetInputForSearchType;
     If search_by_number {
          GetInputNumber;
          PlaySong(song[InputNumber]);
     }
     Else {
          GetInputTitle;
          For all songs stored:
               MatchScore(inputTitle, songTitle[i]);
          PlaySong(song[HighestMatchScoreIndex]);
     }
```

```
int main() {
     while (TRUE) {
          If select = 1:
               CreateSong();
          Else If select = 2:
               PlaySong();
          Else If select = 3;
               ListSongs();
          Else: Invalid Input;
     }
     return 0 ;
}
```

The code follows a specific algorithm for taking in song strings and playing them. The general algorithm is as follows:

1. Pass a song string to StoreSong() function
    a) For every character, check if they are null. If it is, stop and append a R0 note to the song.
    b) If able, store three values as the note letter, and possibly two duration characters. There is a chance that the duration is double digits.
    c) If the value stored in the 2nd digit's place is not actually an integer, the first duration value is the entire duration.
    d) If the value stored in the 2nd digit's place is an integer, the duration has a tens and a ones place.
    e) Pack the notes (see step 2)
    f) Continue steps 1.a. through 1.d. until an "R0" is found or a "\0" null character is found. Jump to step 3.

2. Pass note characters and durations to the PackNote() function
    a) If the character is 'R' or 'r', store 0b00000111 into a binary value holder
    b) Assuming the input is always valid, subtract 65 from the ASCII value because 65 is the offset of ASCII value to the binary bit representation.
    c) Take the result from step 2.a. or 2.b. and shift it to the left by 5 bit places.
    d) Add on the duration value. Return to step 1.f.

3. Play the song using the PlaySong() function.
    a) For respective notes A through G, store a specific frequency value and period value that will be used to make the buzzer produce a specific sound. If there is a rest, R, do not produce sound.

b) Play the note for the specified duration time. The duration time should be quarters of a seconds. If 0b00000001 stored, this means that the sound or rest is held for 0.25 seconds long.
c) Repeat steps 3.a and 3.b for every note in the song.

This project taught me about the memory limitations of some microcontrollers. I also learned about how playing noises at different frequencies and periods can create specific sounds. I have become acquainted with basic c programming, implementing arrays and memory distribution on microcontrollers.