CMPE 311, Fall 2013

Homework 2: Song Composer Using Terminal

Dr. Robucci

October 2, 2013

Summary

Your assignment is to create a terminal-based program that allows a user to enter songs, along with a title and author, and play them back. You will interact with the AVR Butterfly through a terminal program running on a PC. **NOTE: UART code in C will be provided.**

1 Main Menu

Upon boot-up, the menu seen in Figure 1 should appear.

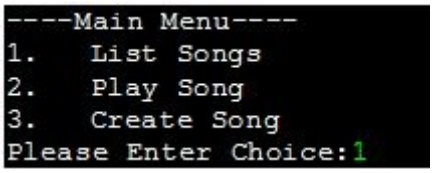

Figure 1: Main Menu

1.1 Main Menu Option 1: List Songs

If option 1 is selected from the main menu, a list of stored songs should be shown (see Figure 2). Then the user should be returned to the main menu.

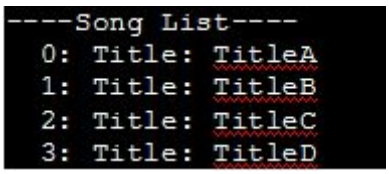

Figure 2: List Menu

1.2 Main Menu Option 2: Play Song

If option 2 is selected from the main menu, the user should is given a menu to choose a song by either searching titles or providing the song number(see Figure 3). After the song is selected, the song number and title should be printed. The program should then return to the Main Menu.

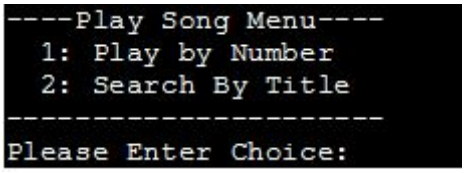

Figure 3: Song Menu

If option 1 is selected, the user should be prompted for a song number corresponding to the number found in the list menu (Section 1.1).

If option 2 is selected, the user should be prompted for the title.

```
Enter title search string (use space between words): hot buns
```

1.2.1 Search Behavior

For option 2 in the Play Song Menu, the song selection should be done as follows:

- Treat the user input string a series of whitespace-separated tokens called the query string.
- Treat the stored titles as series of whitespace-separated tokens called the database string.
- Determine if a matching token is found in the database string for each of the tokens in the query string.
- Count matches for repeated tokens in the query string, **but not repeated tokens in the database string**.
- Create a function to find the number of query tokens that match a token in a database string.
- Perform all matching as case-insensitive.
- Automatically select the song with the **BEST** match and continue by printing the song information (number and title) and playing the song.

Example function: `MatchScore( <query string (user)>, <database string (stored)>);`

```
MatchScore("hello there", "Hello")                          returns 1
MatchScore("hello there", "Hello hello")                    returns 1
MatchScore("hello hello there", "Hello")                    returns 2
MatchScore("hello hello there", "Hello hello")              returns 2
MatchScore("hello hello there", "Hello hello theRe")        returns 3
```

1.3 Main Menu Option 3: Create Song

If option 3 is selected from the main menu, the user should be able to create a song. This does not change the number of stored songs, that will be fixed to 4. This function simply allows overwriting of song 0,1,2 or 3 (user choice).

The user should be prompted for a song title:

```
Enter Title of Song (may include spaces): Hot Cross Buns
```

and then the song itself:

```
Enter Song ABCDEFGR followed by quarter seconds: B2A2G3R1B2A2G3R1
```

The user-input encoding for the song is as follows:

`<letter><quarters><letter><quarters><letter><quarters><letter><quarters>`

- The letter may be uppercase or lower case ABCDEFG or R
- A,B,C,G,D,E,F,and G correspond to tone to be played
- R corresponds to a rest (silence)
- Quarters should be a number from 0 to 31 representing the number of quarter-seconds to play the note or rest.
- The user simply presses enter to finish.
- Once the user enters the song, it must be stored more compactly in order to save memory. Each letter-number pair should be stored as a single byte, using the following encoding:

A 0b000
B 0b001
C 0b010
D 0b011
E 0b100
F 0b101
G 0b110
R 0b111

- The letter encoding should be stored in the upper 3 bits of each byte
- The quarters value should be stored in the lower 5 bits of each byte
- The end of the stored song must be indicated with a zero-length-rest, which is just 0b11100000
- The user does not need to provide R0 to indicate the end of the song, you must append that yourself to the of the store, compacted song array.

1.3.1 Example 1

For the user-provided string: B2A2R1C2
The song stored should be:
0b00100010
0b00000010
0b11100001
0b01000010
0b11100000 –> terminating note automatically appended

1.3.2 Example 2

For the user-provided string: B2A2R1C2R0C2R1
The last two notes are omitted. The song stored should be:
0b00100010
0b00000010
0b11100001
0b01000010
0b11100000 –> terminating "note." After this I don't care
0b01000010 I don't care if the notes following an R0 are stored or not
0b11100001
0b11100000 Likewise, I don't care if you append an R0 or not if the user already provided one

2 Necessary include files for this project:

```
#include <avr/io.h>
#include <util/delay.h>

#include <inttypes.h>
#include <ctype.h>

#include <stdio.h>
#include <string.h>
#include "UO_UART.h"    /// to direct the standard I/O to AVR UART -- UART code in C will be prov
```

3 Useful debug macros

```
#ifdef MYDEBUG
#define DEBUG_PRINTF(...) printf(__VA_ARGS__)
#define DEBUG_PRINTENTER(...) printf("Entering Function %s\n",__func__)
#define DEBUG_PRINTEXIT(...)  printf("Leaving  Function %s\n",__func__)
#else
#define DEBUG_PRINTF(...)
#define DEBUG_PRINTENTER(...)
#define DEBUG_PRINTEXIT(...)
#endif
```

4 Functions/Procedures You Must Write

You will lose points if you do not write and use the following functions in your code.

4.1 Find and replace CR or LF

```
void StripEOL(char string[],int n);
```

StripEOL looks through up to n characters of string searching for '\n' or '\r.' Once it is found, the search should stop and the character should be replaced with '\0' The user input string can be passed through this before further process.

4.2 Menu Function

```
uint8_t DisplayMenu(const char menu[]);
```

DisplayMenu displays a menu, prompts user for input, and returns selection. The parameter menu should be string of '\n'-delimited substrings. The first one is the title and the remaining ones correspond to selection choices. The function must check user input. If an invalid selection is made the user should be reprompted.

Example: userChoice = DisplayMenu(Main Menu\nCreate Song\nPlay Song\nList Songs\n); in this case userChoice should be assigned a value from 1 to 3.

4.3 List Songs Function

```
void ListSongs(char songTitle[NUMBER_OF_SONGS][MAX_TITLE_LENGTH]);
```

ListSongs prints an organized list of songs.

4.4 Play Song Function

Must be finished for week 1 submission.

```
void PlaySong(uint8_t song[]);
```

For week one, at least implement printing the ascii letter and number such as this:∗A2∗∗B2∗∗R2∗

PlaySong plays stored song with ending indicated by 0. Each byte should be 3 bits for note letter and 5 bits for duration. The notes can be 0-6 for notes A-G or 7 for a REST (as seen in Section 1.3) The duration is 0 through 31 and corresponds to the number of quarter-seconds of duration for the note or silence. The end of the string is indicated by a zero duration of silence (0b11100000).

4.5 Pack/Unpack Note Functions

Must be finished for week 1 submission.

```
uint8_t PackNote (char letterASCII, uint8_t duration);
```

PackNote takes input provided in songString and packs information into song bytes as described in Section 1.3. Note that the letter is not stored as ASCII.

```
uint8_t UnpackNoteLetterASCII   (uint8_t packedNote);
```

UnpackNoteLetterASCII returns ASCII character of packed note.

```
uint8_t UnpackNoteDuration (uint8_t packedNote);
```

UnpackNoteDuration returns duration of packed note.

4.6 Song Store Function

Must be finished for week 1 submission.

```
void StoreSong(uint8_t song[],const char songString[]);
```

StoreSong takes ASCII input provided in songString and packs information into song bytes. SongString should be note-duration pairs such as B2A2G3R1B10R0A20G30. Lower-case notes should also be accepted. R corresponds to silence. Any zero-length-silences R0 indicate the end of the song. Notes following an R0 should not be stored. Storage format must be compatible with PlaySong function and adhere to other descriptions in this document (see Section 1.3 for more detail).

4.7 Play Note Functions

Must at least print for week 1 submission.

```
void PlayNote(uint8_t letterASCII, uint8_t quarters);
```

For week one, at least implement printing the ascii letter and number such as this: ∗A2∗

You'll need functions to play notes for variable duration. Here are example constants for a G note:

```
#define FREQ_G4_HZ 392
#define HALFPERIOD_G4_US 1276 //delay in microseconds
```

See http://www.phy.mtu.edu/∼suits/notefreqs.html for frequencies of other notes.

Here is play note pseudo-code:

1. Declare integers `numIterations` and `halfPeriodMicroseconds`
2. Based on the ASCII letter, assign `halfPeriodMicroseconds` and `numIterations`. The number of iterations can be computed from the period and delay as (freq/4)∗quarters
3. Repeat this for `numIterations` times:
   {
   set port B pin 5 low
   delay for halfPeriodMicroseconds
   set port B pin 5 high
   delay for halfPeriodMicroseconds
   }

4.8 Matching Function

```
int MatchScore(const char countQueryString [],const char database[]);
```

MatchScore treats the contents of countQueryString and database as whitespace-delimited tokens. It returns a count of the number of tokens from countQueryString that can be found at least once database. All queries should be treated as case-insensitive. For more information see Section 1.2.1.

5 Useful Stuff for Top of Code

```
#define PORTB5_SPEAKER_MASK 0b00100000

#define USER_LINE_MAX 128
char userLine[LINE_MAX];

#define NUMBER_OF_SONGS 4
#define MAX_SONG_LENGTH 64
#define NOTE_A 0
#define NOTE_B 1
#define NOTE_C 2
#define NOTE_D 3
#define NOTE_E 4
#define NOTE_F 5
#define NOTE_G 6
#define NOTE_R 7

char songTitle [NUMBER_OF_SONGS][STR_LENGTH]={"Title1","Title2","Title3","Title4"};
char song[NUMBER_OF_SONGS][MAX_SONG_LENGTH] =
{{(NOTE_B<<5)+2 ,(NOTE_A<<5) +2,(NOTE_G<<5) +2 },{NOTE_R<<5},{NOTE_R<<5},{NOTE_R<<5}};
    // some initial song with notes

const char menuMain[] = "Main Menu\nCreate Song\nPlay Song\nList Songs\n";
const char menuPlay[] = "Play Menu\nSearch By Title\nNumber\n";
```

See Discussion 6 notes for how to store constant strings in program memory.

6 User I/O from terminal

Add this to code outside main or other function:

```
include "U0_UART.h"    /// to direct the standar I/O to AVR UART -- UART code in C will be provid
extern FILE uart_stream;   // defined in U0_UART.c -- stream that
                           //scanf and printf will use
```

Add this to main:

```
  stderr = stdout = stdin = &uart_stream;  /* Redirecting the standard I/O */
  UARTInit();    /* Initial UART */
```

Example to grab number:

```
    if (fgets(userLine,10,stdin)!=NULL) {
        sscanf(userLine,"%d",&userNum);
    }
```

Example to grab string with spaces into row of 2-D array:

```
  //songTitle is a 2D char array and userLine is a char array
  fgets(userLine,LINE_LENGTH_MAX,stdin);
  stripEOL(userLine,LINE_LENGTH_MAX);
  strcpy(songTitle[userNum],userLine);
```

7 Week 1 Submission

Implement:

- `uint8_t PackNote (char letterASCII, uint8_t duration);`
- `uint8_t UnpackNoteLetterASCII   (uint8_t packedNote);`
- `uint8_t UnpackNoteDuration (uint8_t packedNote);`
- `void StoreSong(uint8_t song[],const char songString[]);`
- `void PlayNote(uint8_t letterASCII, uint8_t quarters);`

  (For week 1, create a print version that prints the note and duration. e.g. PlayNote(A,2) prints ∗A2∗)
- `void PlaySong(uint8_t song[]);`

  (For week 1, create a print version that prints the note and duration. e.g. PlaySong(songPacked) prints ∗A2∗∗B2∗∗R2∗)

Code such as following should work:

```
char songASCII[] = A2B2R2;  //a valid user input string
uint8_t songPacked [10];
void StoreSong(songPacked,songASCII);
void PlaySong(songPacked);
```

Prints: *A2**B2**R2*

Submit the week1 report, and library functions with a music.h file and a music.c as a tar file using the following command.

submit cmpe311 hw2_week1 <glusername>.tar

8 Final Submission

Submit the final report, Makefile, and library functions with a music.h file and a music.c as a tar file using the following command.

submit cmpe311 hw2 <glusername>.tar

Final report must include a usage manual, project description, and any ambiguities you resolved. Separate final report submissions can be made using the following command.

submit cmpe311 hw2_report <report name>.pdf