C Programming lab
Assignment 2

Revision Date: September 3, 2014

Music!

"He who hears *music*, feels his solitude peopled at once" - Robert Browning

Your task is to develop a filter to remove the initial white noise in an audio file and then play the resulting song. First create a directory named *music* that hangs off your *clab* directory. All your work will be in this directory. So make sure, you're in the right directory by running the following linux command

```
pwd
```

Getting the necessary bits

Download *arplay*, *rplay*, and *song.rra* files from either LMS account or the dropbox account. *arplay* and *rplay* are the executables files and you need to change the permission values before using these files as shown below.

```
chmod a+x arplay;
chmod a+x rplay;
```

Playing the song

You can play the audio file *song.rra* by using *arplay* or *rplay* utility tools. But before testing the *arplay* and *rplay* tool, make sure that you close any other audio plugins running in your browser. For example youtube web page should be closed. You can then run the audio file using the tool as shown below.

```
./arplay song.rra
```

If you get any error saying *Device or resource busy*, try *rplay* instead of *arplay.*

```
./rplay song.rra
```

Make sure that your sound settings is not in mute position. Note that there is an initial silence/noise of 3 seconds!! Your task is to write a C program in *play.c*to automatically remove this initial silence. Your program will also incorporate a threshold value, which is discussed in the next section.

Input/Output

The threshold value and the song to be stripped of its initial silence are to be supplied as command line arguments, as follows:

```
./play <thresholdValue> <inputFile>
```

The `<thresholdValue>` command-line argument is an integer value which is used to remove initial data values whose absolute values are below the given threshold. The reason for the threshold is that silence is rarely the absence of sound, but rather a very low sound level. Note that the term 'absolute value' indicates that any value between the *positive* or *negative* threshold value is ignored during the initial audio data scanning. The `<inputFile>` command-line argument specifies the file name of the audio file you wish to process. Here is an example call to the program.

```
./play 20 song.rra
```

This strips out all the initial data until a value greater than 20 or less than -20 is encountered. The output of the play program is sent to the screen. The next command sends the output to the arplay utility, which plays the song:

```
./play 20 song.rra | ./arplay
```

If *arplay* does not work, try *rplay*.

Instead of playing the output, you can save it to a file. As an example, the command:

```
./play 20 song.rra > new.rra
```

saves the processed audio in a file named *new.rra*. You can look at both the original file and the newly saved file using *vim* and you can play the new file using *arplay*:

```
./arplay new.rra
```

RRA file

RRA stands for *readily readable audio*. It is designed to be easily viewed (and debugged) and to be extensible. Originally suggested by a student named Ian Taylor for use in a High School Programming Contest, it has been adopted by Professor J. C. Lusth in his music research. The RRA file is composed of two sections, a header section and a data section. The header of the RRA file is as follows:

```
RRAUDIO
...
%%
```

The header must start with the token `RRAUDIO`, while the token `%%` indicates the end of the header. The RRA sound data follows the header as shown below:

```
RRAUDIO
%%
0
0
1
-2
...
```

In between the RRAUDIO token and the %% token, information about the audio file may appear as shown below.

```
RRAUDIO
createdBy: songlib
samples: 000002345102
modifiedBy: wnoise
%%
0
0
1
-2
...
```

Your program is supposed to print the exact text in header to the output console. If any header information is lost, the audio file won't run.

All command-line arguments are strings, so you will need to convert the threshold value argument to an float using *atof* function. That is to say, you must know how to convert a string of digits:

```
"150"
```

into a number:

```
150
```

A program that fails with a syntax or runtime error on the *song.rra* will be given a score of zero. Note that your program will be tested with different audio RRA files with different threshold values to check whether your code is thoroughly tested. So students can create their own different RRA files and test their output.

Code

Your source-code should be properly indented. If not, significant points will be deducted. The variable names should be readable. Points will be deducted if you use the variables such as *x, y, z, i, j, k*. Only for loop variables, you are allowed to use one-letter variable name. For example, if a variable is used to hold audio sample values, you can use variable name as *sampleVal*.

Plaigarism

Your code will be checked with other students code in the class to detect the percentage of plaigarism. Moss (Measure of software similarity) software tool from Stanford University, an automatic system for determining the similarity of programs will be used. If the percentage of plaigarism is found above *90%*, you will lose all your points in this assignment.

Submission Instructions

Change to the directory containing your assignment *music*. Do an *ls* command. You should see something like this:

```
    play.c song.rra new.rra play
```

among other files.

You can now submit using the following command:

```
    submit clab mr music <your-iiitb.org-email-address>
```

Make sure that you are in *music* directory and you do the submission, otherwise you will lose 100 points. Students can develop their code on their personal laptops, but compile your code in your own laboratory system and check whether your executable is running before submission. If the executbale is not running at instructor's end, you will lose all points.

Due Date

The due date for this assignment is 24th September, 11:59:59 AM.