ComS 229 Project 1 (Part 2)

Tools for digital sound

September 15, 2013

0 Introduction

This is a "change" document. In other words, rather than writing another entire spec, this document focuses on what should be changed relative to part 1 of the project.

1 The programs

Write or update the following programs, in ANSI C. As usual, this describes the *minimum* required functionality; you are free to implement extra if you like. For example, you might want to add a `--debug` switch to turn on debugging messages. All switches for this project will use a single dash, so using two will ensure that you do not choose a switch that will be added later.

1.0 `sndinfo`

In this version of `sndinfo`, you must support passing the filenames and other switches as arguments. Specifically, arguments may be passed as:

```
$ sndinfo [switches] [file] [file] ...
```

where brackets indicate optional arguments. For each specified input file, `sndinfo` should display the information for the file, in the same format as before, except there should be only one "line of dashes" in between the file information (see Figure 1). If no filenames are specified, then `sndinfo` should treat the standard input stream as the file, and show a filename of "`(standard input)`". You must support the following switches:

-h : Display a short help screen to standard error, and terminate cleanly without reading any files.

-1 : Behave as specified in part 1 of the project (i.e., prompting for filenames and ignoring arguments). When grading your code for part 1, this switch will be used, so you may start working on part 2 before part 1 is graded. Note that if this switch is not used, then `sndinfo` should not prompt for *anything*.

1.1 `sndconv`

In this version, `sndconv` will read from standard input and write to standard output. I.e., the standard input stream should be treated as the input file, and the converted file should be written directly to the standard output stream. Any messages (e.g., errors) should be written to standard error. You must support the following switches, passed as arguments.

-h : Display a short help screen to standard error, and then terminate cleanly.

```
$ ./sndinfo hello.cs229 zep.aiff
-------------------------------------------------------------
   Filename: hello.cs229
     Format: CS229
Sample Rate: 22050
  Bit Depth: 16
   Channels: 1
    Samples: 13339
   Duration: 0:00:00.60
-------------------------------------------------------------
   Filename: zep.aiff
     Format: AIFF
Sample Rate: 44100
  Bit Depth: 16
   Channels: 2
    Samples: 529995
   Duration: 0:00:12.02
-------------------------------------------------------------
$
```

Figure 1: Example output for `sndinfo`.

-1 : Behave as specified in part 1 of the project (i.e., prompting for filenames and ignoring arguments). When grading your code for part 1, this switch will be used, so you may start working on part 2 before part 1 is graded. Note that if this switch is not used, then `sndinfo` should not prompt for *anything*.

-a : Output should be AIFF, regardless of the input format.

-c : Output should be CS229, regardless of the input format.

1.2 `sndcat`

This program reads all sound files passed as arguments, and writes a single sound file where the sample data is the concatenation of the sample data in the inputs. The resulting sound file should be written directly to standard output. If no files are passed as arguments, then the standard input stream should be treated as the input file. Any messages should be written to standard error. You must support the following switches, passed as arguments.

-h : Display a short help screen to standard error, and then terminate cleanly.

-a : Output should be AIFF.

-c : Output should be CS229.

1.3 `sndcut`

This program reads a sound file from the standard input stream, and writes the sound file to the standard output stream in the same format, after removing all samples specified as arguments. Any messages should be written to standard error. Specifically, arguments may be passed as:

```
$ sndcut [switches] [low..high] [low..high] ...
```

where `low..high` specifies that all samples numbered between `low` and `high`, inclusive, are to be removed. You must support the following switches, passed as arguments.

-h : Display a short help screen to standard error, and then terminate cleanly.

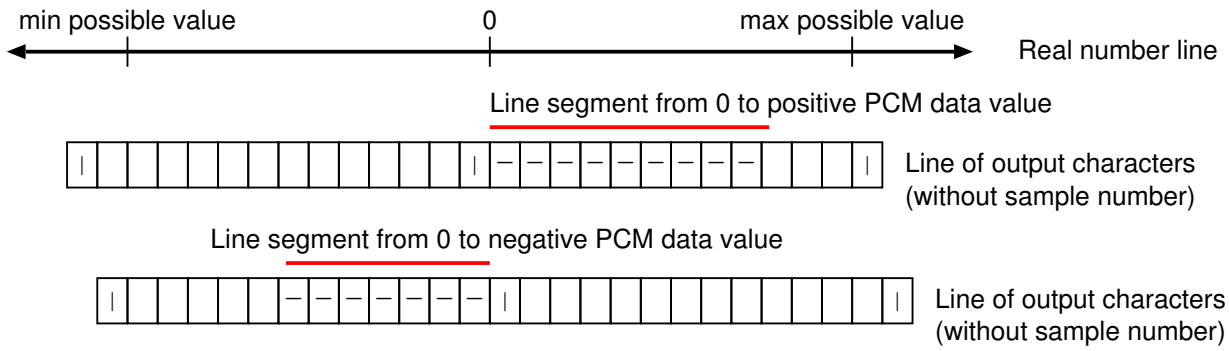

Figure 2: Illustration of how sample data should be "plotted".

1.4 `sndshow`

This program reads a sound file from the standard input stream, and displays an "ASCII art" representation of the sample data. Your output should be formatted as follows. The first 9 characters are used to display the sample number which should be "right justified". Next comes a pipe character ("|"). The next x characters are used for "plotting" negative values. Next comes a pipe character that represents the value 0. The next x characters are used for "plotting" positive values. The final character is another pipe character. The value of x is dictated by the desired total output width; see the program switches.

For each sample frame, there will be c lines of output, where c is the number of channels. The sample number should be displayed only for the first line of output for the sample. Then, for each channel, you should draw a line using the dash character to represent the sample value for that channel. Conceptually, the line should be drawn as shown in Figure 2: draw a line segment between 0 and the sample value, then overlay x equal–width boxes between 0 and the maximum sample value, if the sample value is positive, or between 0 and the minimum sample value, if the sample value is negative. For each box that is half full or more, write a "-" character, otherwise write a space.

You must support the following switches, passed as arguments.

- **-h** : Display a short help screen to standard error, and then terminate cleanly.
- **-c** c : Show the output only for channel c, for $1 \leq c \leq$#channels.
- **-w** w : Specify the total output width, in number of characters. If not specified, the default is $w = 80$. Note that w must be even so that the number of characters for representing positive values is equal to the number of characters for representing negative values. You must support values down to $w = 20$.
- **-z** n : Zoom out by a factor of n. If not specified, the default is $n = 1$. The value to "plot" should be the largest *magnitude* value over n consecutive samples, and the number of lines of output should decrease by about a factor of n.

Example output for a tiny sound file is shown in Figure 3.

2 Submitting your work

You should turn in a gzipped tarball containing your source code, makefile, and a `README` file that documents your work. The tarball should be uploaded in Blackboard.

Your executables will be tested on `pyrite.cs.iastate.edu`. You should **test early, and test often on `pyrite`**. We will use some scripts to check your code; this means you should not change the name of the executables or the order in which your programs prompt for input.

3 Grading

The following distribution of points will be used for this part of the project.

sndinfo : 50 points

sndconv : 50 points

sndcat : 150 points

sndcut : 150 points

sndshow : 150 points

makefile : 30 points

As before, typing "`make`" should build all of your executables, and "`make clean`" should remove the executables and any object files.

Documentation & style : 30 points

Based on the `README` file only.

Total part 2 : 610 points

```
$ cat tiny.txt
CS229
Channels 3
BitDepth 8
SampleRate 1000
StartData
0   -127 127
10  45   -103
20  83   -4
30  0    99
40  44   45
$ ./sndshow -w 32 < tiny.txt
        0|          |          |
         |----------|          |
         |          |----------|
        1|          |-         |
         |          |----      |
         |  --------|          |
        2|          |--        |
         |          |-------   |
         |          |          |
        3|          |--        |
         |          |          |
         |          |--------  |
        4|          |---       |
         |          |---       |
         |          |----      |
$ ./sndshow -w 32 -z 3 < tiny.txt
        0|          |--        |
         |----------|          |
         |          |----------|
        3|          |---       |
         |          |---       |
         |          |--------  |
$ ./sndshow -w 72 -z 3 < tiny.txt
        0|                              |-----                         |
         |------------------------------|                              |
         |                              |------------------------------|
        3|                              |---------                     |
         |                              |----------                    |
         |                              |-----------------------       |
$ ./sndshow -w 72 -z 3 -c 1 < tiny.txt
        0|                              |-----                         |
        3|                              |---------                     |
$ ./sndshow -w 72 -z 3 -c 2 < tiny.txt
        0|------------------------------|                              |
        3|                              |----------                    |
$
```

Figure 3: Example output for `sndshow`.