CS 330: Homework 4
Nullaby

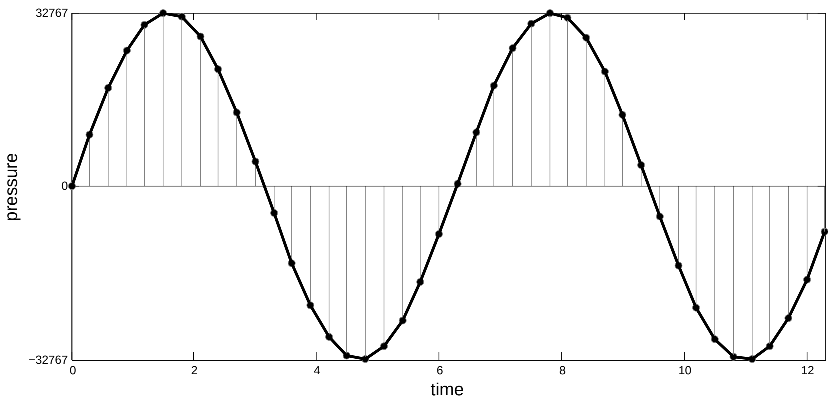
1 Description

Your goal in this homework is to expand your awareness of the beauty and pain that is C programming combined with the manipulation of XML-structured data. You will do this in the context of writing a utility to convert an XML music file into a WAV file.

2 Background

2.1 Sound

Sound is a wave of pressure that propogates through the physical world. The frequency of the wave determines its pitch. Each note on the musical staff is associated with a particular frequency, and to create a sound clip at the frequency, we walk along the wave modeled by $\mathit{amplitude} \cdot \sin(2\pi \cdot \mathit{frequency} \cdot t)$, where t represents our position on the time axis. We capture the magnitude of the wave at all samples t, and this sequence of samples is our digitized sound clip; it's what we store in the WAV file.



As this homework is not so much about digitizing sound, we have provided some helper functions to map notes to frequencies and to generate a sequence of samples for a given frequency and duration. You will use these functions (or you can write your own) to turn a textual representation of some music into a WAV file.

Please have a look through the helper functions before reading on:

`http://www.twodee.org/teaching/cs330/2012A/homework/helpers.c`

As a class, we'll walk through the WAV format. It is described in detail here:

`https://ccrma.stanford.edu/courses/422/projects/WaveFormat/`

2.2 XML Music

The musical notation for this homework is a simple XML format. The following example demonstrates all its features:

```
<?xml version="1.0" encoding="UTF-8"?>
<song octave="4" duration="2">
 <note name="C"/>
 <note name="D+" duration="1"/>
 <rest duration="2"/>
 <repeat times="3">
   <note name="B-" octave="5"/>
 </repeat>
</song>
```

You don't need to have any background in music to be able to complete this assignment, though you'll have to doggedly follow the specification of the helper methods and see how the XML file and the helper methods fit together.

The following rules describe the XML file structure in more detail. You may assume the XML file is well-formed, but you cannot assume the attributes to be in any particular order.

1. The root entity is `song`, which you may assume has a default octave and note/rest duration specified in its attributes. Durations are of the form 1 (a whole note), 2 (half), 4 (quarter), 8 (eighth), 16 (sixteenth), or 32 (thirty-second).
2. Nested in `song` is a sequence of `note`, `rest`, and `repeat` entities.
3. All `note`s have a `name` attribute, one of A, B, C, D, E, F, G, optionally followed by a + (sharp, raise the note a half step) or − (flat, lower the note a half step). Notes may or may not have `octave` and `duration` attributes. If they do not, the containing `song`'s attributes are used.
4. `Rest`s may or may not have a `duration` attribute. If they do not, the containing `song`'s `duration` attribute is used. Rests effectively have a frequency of 0.
5. Repeated sequences are grouped in a `repeat` entity. You may assume such entities have a `times` attribute indicating how many times the sequence is to be repeated. Repeated sequences do not nest.

3 Requirements

To receive full credit for this assignment, you must satisfy the following requirements:

- Create a directory named `nullaby` and place all of your files inside of it.

- Write your code in `nullaby.c`. No header file is specified.
- Have a `main` function that expects command-line arguments for the input XML file and a path to the output WAV file. Your code must transform the music described in the XML to the binary WAV file. No other specification is given, but please keep your functions short, document them using Javadoc syntax (see the `dirch` example on Piazza), and use meaningful names.
- Parse the XML file with `libexpat`. Use no global variables. Use `XML_SetUserData` to pass data around to your callbacks.
- Create a `makefile` whose default/first rule builds your executable. Add a `clean` rule that deletes your executable.
- Have no memory leaks.
- The makefile and the `nullaby` executable must work on `clark.cs.uwec.edu`.

4 Advice

You do not know your memory needs ahead of time. Do not try to guess its size ahead of time. Instead, grow your samples buffer with each new note or rest with the following algorithm:

```
make new buffer big enough for song_so_far plus the new clip
copy in current samples
append new clip
song_so_far = new buffer
```

5 Submission

Before submitting, you are encouraged to write test code and share it with the class on Piazza. Individuals who do so will have their names inserted into a lottery for a fabulous prize to be awarded at the end of the semester. You are especially encouraged to share your WAV files and XML compositions.

Please submit according to these instructions. Violators will be prosecuted.

1. One level above your `nullaby` directory, run the command `zip -r nullaby.zip nullaby` to create a ZIP archive of your files.
2. Drop your `nullaby.zip` file into your submission directory on the `W:` drive. You may overwrite this file as often as you like before the deadline.