This is CS50x

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan)
(https://www.tiktok.com/@davidjmalan) (https://twitter.com/davidjmalan)

Lab 4: Volume

> You are welcome to collaborate with one or two classmates on this lab, though it is expected that every student in any such group contribute equally to the lab.

Write a program to modify the volume of an audio file.

```
$ ./volume INPUT.wav OUTPUT.wav 2.0
```

Where `INPUT.wav` is the name of an original audio file and `OUTPUT.wav` is the name of an audio file with a volume that has been scaled by the given factor (e.g., 2.0).

WAV Files

WAV files (https://docs.fileformat.com/audio/wav/) are a common file format for representing audio. WAV files store audio as a sequence of "samples": numbers that represent the value of some audio signal at a particular point in time. WAV files begin with a 44-byte "header" that contains

information about the file itself, including the size of the file, the number of samples per second, and the size of each sample. After the header, the WAV file contains a sequence of samples, each a single 2-byte (16-bit) integer representing the audio signal at a particular point in time.

Scaling each sample value by a given factor has the effect of changing the volume of the audio. Multiplying each sample value by 2.0, for example, will have the effect of doubling the volume of the origin audio. Multiplying each sample by 0.5, meanwhile, will have the effect of cutting the volume in half.

Types

So far, we've seen a number of different types in C, including `int`, `bool`, `char`, `double`, `float`, and `long`. Inside a header file called `stdint.h` are the declarations of a number of other types that allow us to very precisely define the size (in bits) and sign (signed or unsigned) of an integer. Two types in particular will be useful to us in this lab.

- `uint8_t` is a type that stores an 8-bit unsigned (i.e., not negative) integer. We can treat each byte of a WAV file's header as a `uint8_t` value.
- `int16_t` is a type that stores a 16-bit signed (i.e., positive or negative) integer. We can treat each sample of audio in a WAV file as an `int16_t` value.

Getting Started

> Started CS50x in 2021 or prior and need to migrate your work from CS50 IDE to the new VS Code codespace? Be sure to check out our instructions on how to **migrate** your files!

Open VS Code (https://code.cs50.io/).

Start by clicking inside your terminal window, then execute `cd` by itself. You should find that its "prompt" resembles the below.

```
$
```

Click inside of that terminal window and then execute

```
wget https://cdn.cs50.net/2021/fall/labs/4/volume.zip
```

followed by Enter in order to download a ZIP called `volume.zip` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that

matter!

Now execute

```
unzip volume.zip
```

to create a folder called `volume`. You no longer need the ZIP file, so you can execute

```
rm volume.zip
```

and respond with “y” followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd volume
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
volume/ $
```

If all was successful, you should execute

```
ls
```

and you should see a `volume.c` file alongside an `input.wav` file.

If you run into any trouble, follow these same steps again and see if you can determine where you went wrong!

Implementation Details

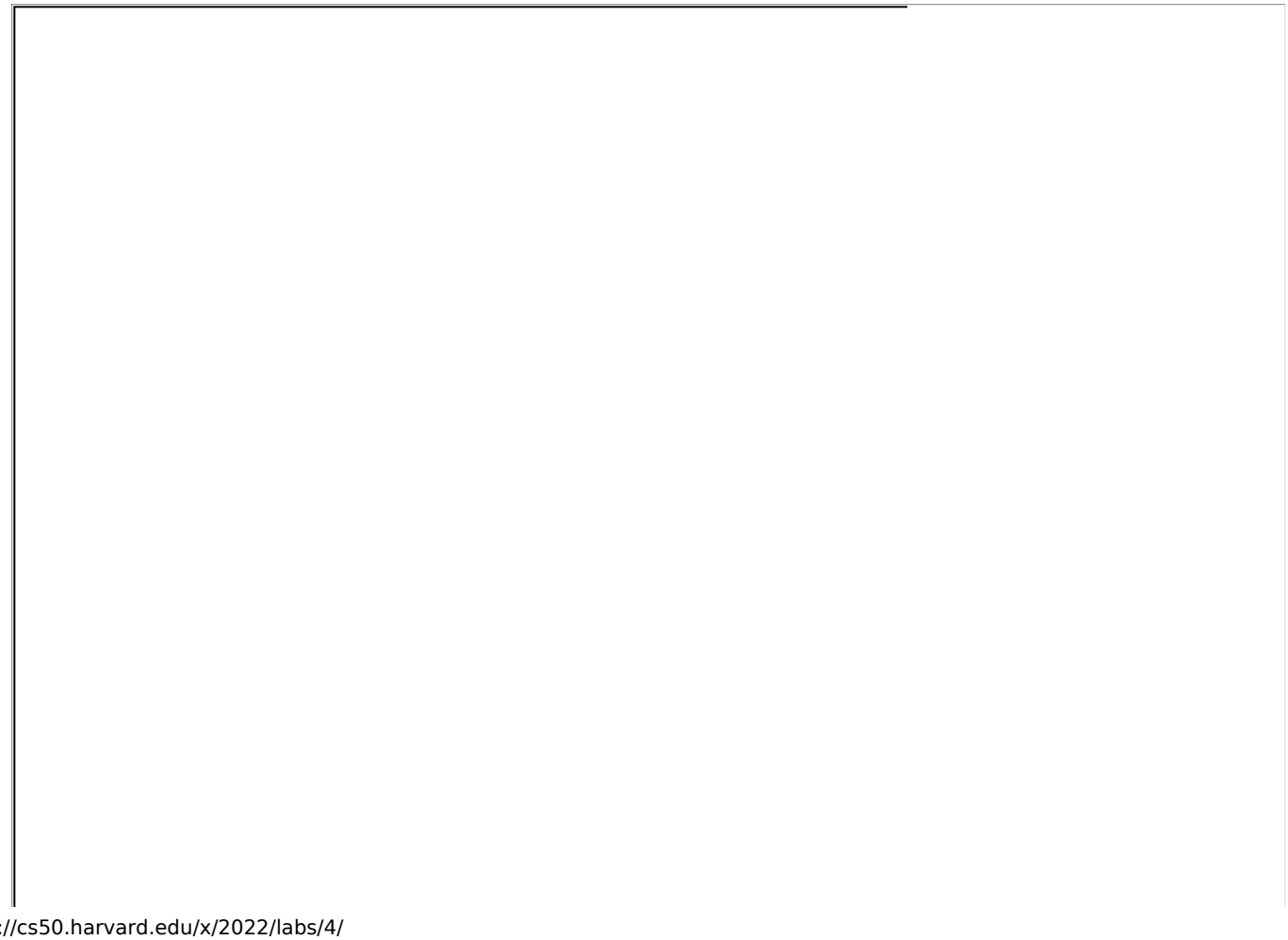
Complete the implementation of `volume.c`, such that it changes the volume of a sound file by a given factor.

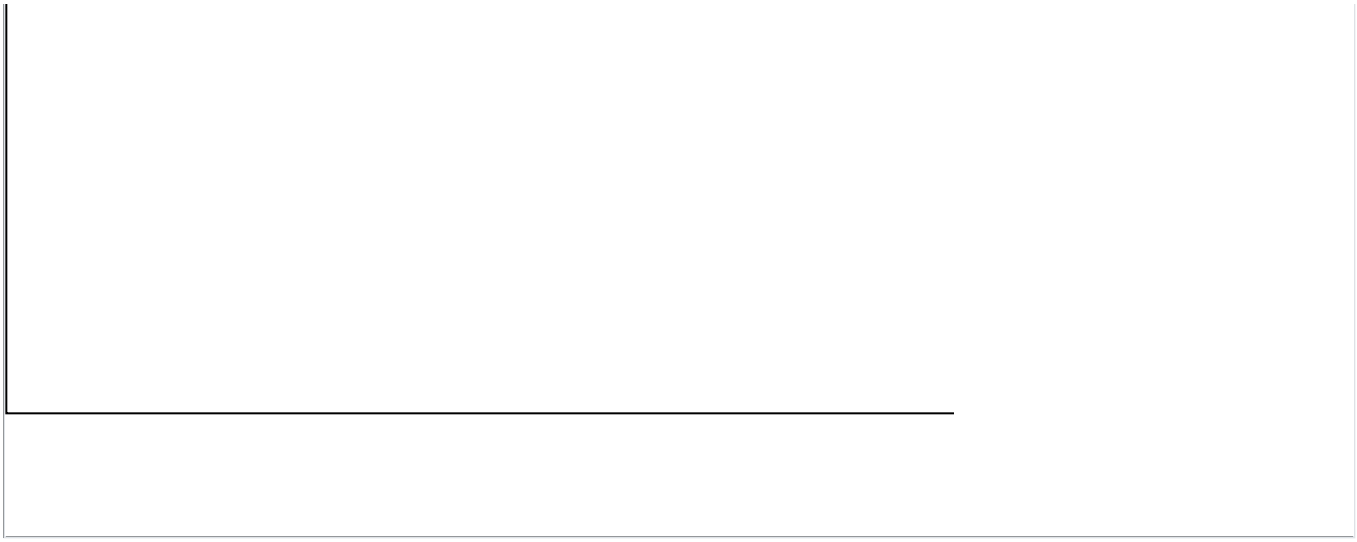
- The program accepts three command-line arguments: `input` represents the name of the original audio file, `output` represents the name of the new audio file that should be generated, and `factor` is the amount by which the volume of the original audio file should be scaled.
    - For example, if `factor` is `2.0`, then your program should double the volume of the audio file in `input` and save the newly generated audio file in `output`.

- Your program should first read the header from the input file and write the header to the output file. Recall that this header is always exactly 44 bytes long.
    - Note that `volume.c` already defines a variable for you called `HEADER_SIZE`, equal to the number of bytes in the header.
- Your program should then read the rest of the data from the WAV file, one 16-bit (2-byte) sample at a time. Your program should multiply each sample by the `factor` and write the new sample to the output file.
    - You may assume that the WAV file will use 16-bit signed values as samples. In practice, WAV files can have varying numbers of bits per sample, but we'll assume 16-bit samples for this lab.
- Your program, if it uses `malloc`, must not leak any memory.

Walkthrough

This video was recorded when the course was still using CS50 IDE for writing code. Though the interface may look different from your codespace, the behavior of the two environments should be largely similar!





Hints

- You'll likely want to create an array of bytes to store the data from the WAV file header that you'll read from the input file. Using the `uint8_t` type to represent a byte, you can create an array of `n` bytes for your header with syntax like

```
uint8_t header[n];
```

replacing `n` with the number of bytes. You can then use `header` as an argument to `fread` or `fwrite` to read into or write from the header.

- You'll likely want to create a "buffer" in which to store audio samples that you read from the WAV file. Using the `int16_t` type to store an audio sample, you can create a buffer variable with syntax like

```
int16_t buffer;
```

You can then use `&buffer` as an argument to `fread` or `fwrite` to read into or write from the buffer. (Recall that the `&` operator is used to get the address of the variable.)

- You may find the documentation for `fread` (https://man.cs50.io/3/fread) and `fwrite` (https://man.cs50.io/3/fwrite) helpful here.
  - In particular, note that both functions accept the following arguments:
    - `ptr`: a pointer to the location in memory to store data (when reading from a file) or from which to write data (when writing data to a file)
    - `size`: the number of bytes in an item of data
    - `nmemb`: the number of items of data (each of `size` bytes) to read or write
    - `stream`: the file pointer to be read from or written to

- `stream`: the file pointer to be read from or written to
- Per its documentation, `fread` will return the number of items of data successfully read. You may find this useful to check for when you've reached the end of the file!

▶ **Not sure how to solve?**

How to Test Your Code

Your program should behave per the examples below.

```
$ ./volume input.wav output.wav 2.0
```

When you listen to `output.wav` (as by control-clicking on `output.wav` in the file browser, choosing **Download**, and then opening the file in an audio player on your computer), it should be twice as loud as `input.wav`!

```
$ ./volume input.wav output.wav 0.5
```

When you listen to `output.wav`, it should be half as loud as `input.wav`!

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/labs/2022/x/volume
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 volume.c
```

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/labs/2022/x/volume
```