

Audio Converter : RAW, WAV, and FLAC

Designer: [Chen Kangrui](#) and [Fan Site](#).

Overview

In this project, you need to implement a library in C/C++ that supports encoding conversion from .pcm raw audio files to `.wav` files and then to simple `.flac` files, as well as decoding wav and flac format files. Also you need to implement a **Command Line Interface (CLI)** to demonstrate your project.

Audio Formats

PCM

A **raw audio file** is any file containing un-containerized and uncompressed audio. The data is stored as **raw pulse-code modulation (PCM)** values **without any metadata header information** (such as sampling rate, bit depth, endian, or number of channels).

WAV

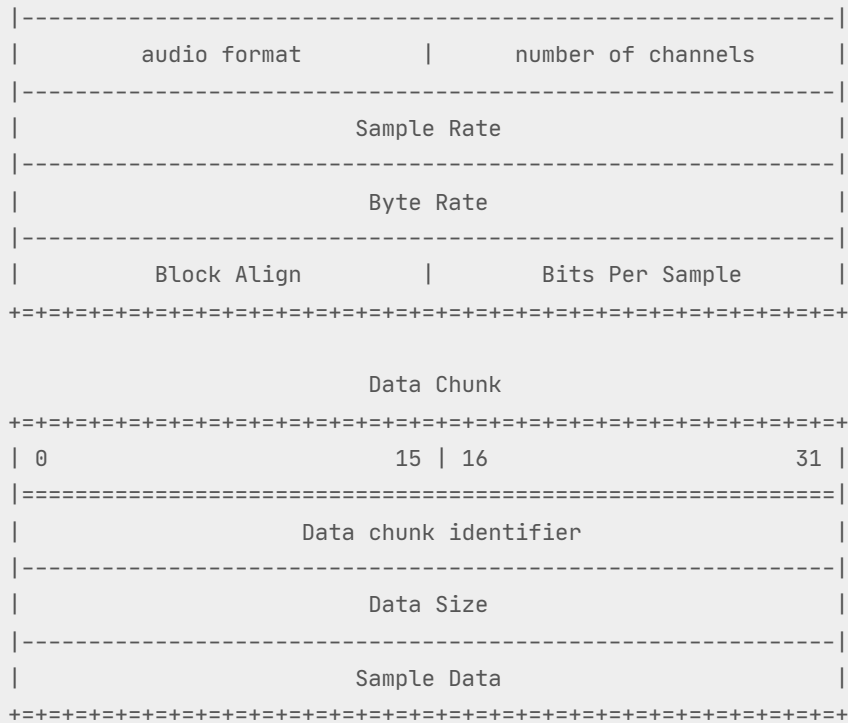
WAV, known for WAVE (Waveform Audio File Format), is a subset of Microsoft's Resource Interchange File Format (RIFF) specification for storing digital audio files. The format doesn't apply any compression to the bitstream and stores the audio recordings with different sampling rates and bitrates. This format is of **little-endian**!

RIFF Header

```
+=====+
| 0                               15 | 16                               31 |
|=====|
|                               RIFF header identifier                     |
|-----|
|                               Total Size                                 |
|-----|
|                               File type                                  |
+=====+
```

Format Chunk

```
+=====+
| 0                               15 | 16                               31 |
|=====|
|                               Format chunk identifier                     |
|-----|
|                               Chunk Size                                  |
+=====+
```



FLAC

FLAC stands for **Free Lossless Audio Codec**, an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality.

FLAC stands out as the fastest and most widely supported lossless audio codec, and the only one that at once is non-proprietary, is unencumbered by patents, has an open-source reference implementation, has a well documented format and API, and has several other independent implementations. This format is of **big-endian**!

More information like the FLAC format specification can be found on the [official documentation of FLAC](#).

Requirements

The tasks below are listed in ascending order of difficulty to implement, and we recommend that you implement them in this order.

Metadata Display & Edit (15 pts)

In this part, you need to implement an executable file to display and edit the metadata of FLAC files. Given a `.flac` file, your program should be able to decode and display the information of metadata blocks of **type 0 (STREAMINFO)** and **type 4 (VORBIS_COMMENT)**.

A `.flac` file is given as the official testcase for this task.

Encoder: RAW → WAV → FLAC (30 pts)

In this part, you need to implement a convertor from `.raw` to `.wav` (10 pts), and then to `.flac` (20 pts).

The first subtask is to convert a raw audio file to a `.wav` file. Since the raw audio file does not contain information like sample rate, bits per sample (depth), number of channel etc., when converting `.raw` to `.wav`, an additional JSON file is needed to provide extra information for the header. In official test cases, a `.json` file is given along with each `.raw` file.

The second subtask is to convert a `.wav` file into a `.flac` file. For basic requirement, the depth will only be **16Bit int** or **24Bit int**, and you only need to encode every subblock into a subframe using **verbatim predictor** with channel assignment of **2 channels: left**, with **STREAMINFO** and **VORBIS_COMMENT** metadata.

An important evaluation metric in this task is whether the audio files converted by your library can be played with commonly used media players.

A `.raw` file with `.json` is given as the official testcase for this task.

Decoder: FLAC → WAV → RAW (30 pts)

For the third task, you are required to implement a decoder for your library, to convert a `.flac` file into `.wav` format (20 pts), and then to `.raw` format (10 pts).

Same as above, you only need to handle simple `.flac` files with **STREAMINFO** and **VORBIS_COMMENT** metadata and **verbatim subframes**.

You can choose the `.flac` file generated in the second task as the input (testcase) for this task, and check if the `.wav` and `.raw` files are the same as the input ones of the encoder.

No official testcase is given for this task.

Possible Bonus Ideas (∞ pts)

Here we offer some ideas about the possible bonus functions, with estimated difficulties. You are also encouraged to explore your own bonus ideas.

The tasks below are listed in ascending order of difficulty to implement.

- 😊 **Excellent Project Management:**

How to manage a project during development can be very easy, or very hard. Try to manage your project structure well, keep a changelog for revision control, write a user-friendly document, cooperate via GitHub, GitLab, Gitee, ...

This should have been a basic requirement, but we set it as bonus to encourage you to learn how to manage a cooperating project well, which is an important skill that you can benefit from in future project development.

Stop using QQ/Wechat to synchronize your project!

That's too brutal!

- 😊 **More Metadata Block Types:**

Support (analyze, display and edit) additional metadata block types, like **PADDING**, **SEEKTABLE**, **PICTURE** etc.

- If a **COMMENT** block is followed by a **PADDING** block, the additional comments to be appended should overwrite part of the **PADDING** block first rather than insert directly.
- If an existing **SEEKTABLE** block (one .flac file has at most one **SEEKTABLE** block) has placeholder seekpoints, the added ones are supposed to replace the placeholders. And if it's followed by a **PADDING** block, do the same as the direction above.

- 😊 **Extreme Robustness:**

Although robustness is one of the basic requirements to obtain basic points, you can get bonus points if your library is excellent in terms of exception handling and memory management!

MD5 signature and CRC check should also be covered in your decoder for this bonus.

However, to get this bonus part, you need to rethink what kind of robustness can be qualified as "**extreme**".

Oops! Why segmentation fault again!

- 🤖 **Interchannel Decorrelation**

Though the basic requirement for channel assignment is to store the left channel and the right channel separately, there are more ways to deal with 2-channel audios, which can be helpful for compression. So try to support at least one of these **special stereo assignments** (from 1000 to 1010) with your convertor.

- To implement this function, you may need to modify both the encoder and decoder, to adapt to different channel assignment.

- 🤖 **More Subframe Types**

Support additional subblock encoding patterns like **SUBFRAME_CONSTANT**, **SUBFRAME_FIXED**, **SUBFRAME_LPC**. You will learn the magic of lossless data compression during this bonus.

- When implementing this part, you are doing **real lossless compression**, the .flac file generated will reduce in size.
- **CONSTANT** is recommended, the other two may need some pre-knowledge of signal processing.

My dad taught me that in Hawaii.

- 🤖 GUI

Implement a pretty GUI for your convertor! However, you should still focus on the main part of this project: your library.

Gold needs to shine, too.

- 🏊 All-round Decoder

FLAC decoder testbench provides a collection of FLAC files that can be used to test various FLAC decoder abilities. Improve your decoder to pass as many testcases as possible [here](#), the more robust your library is, the more bonus points you get.

It's a hexagonal warrior.

- 🧐 More Audio Formats

Modify your library to make it work for more audio formats. A recommended challenge for those who are familiar with the audio file encoding formats.

No pain, no gain.

- Anything You Regard as "Bonus"

Your team is encouraged to come up with your own ideas, we will grade your bonus according to your workload and implementation.

Please describe your bonus clearly in your report, so that inspectors can fully understand and grade your bonus ideas.

Self-Prepared Testcases (5 pts)

You are required to prepare at least 1 testcase for each basic function.

For your bonus part, you should prepare at least 1 testcase for each bonus function, otherwise the bonus is invalid.

Good news: any valid testcases that check the full usability of your library will obtain all of the 5 points.

Report (10 pts)

The report should be in **PDF** format, easy to understand and provide a clear description of the project, especially the **highlights**. Notice that your report should contain everything that a good README has.

The report should also include a detailed description of each team member's contribution, and contribution rated as

- A (significant contribution)
- B (moderate contribution)
- C (little contribution)
- D (free rider).

Reports that meaninglessly pile up pages will result in lower scores. Please be as concise and brief as possible and just show the highlights. Please avoid pasting too much code in your report. Any non-technical detail should be discarded, that's a waste of time for both your team and the inspectors.

You are encouraged to use IEEE, ACM, or any other elegant templates, however this will not affect your score, just to ease the veins of inspectors.

Grading Policy

The maximum of this project is 110 points, consists of:

- basic functions (75)
- report (10)
- self-prepared testcases (5)
- bonus functions (20 + overflow to make up for basic requirements)

1. Your library need to support basic functions, including metadata display and editing, encoding and decoding, and pass all official testcases.

After implementing the basic part, the **upper bound** of your basic score will reach 75, failure to pass the basic usability test will make your upper bound less than 75.

2. If your bonus part gains over 20 points, the overflow part can fill the basic part loss.
3. Be careful of the memory management and exception handling! Your library should be robust enough for basic use. Bad memory management will reduce your score.
4. Attention should be paid to code style. Adequate time is given for code to be written correctly and with good style. Deductions on the score will be made for poor code style. Code style guides, such as the [Google C++ Style Guide](#), can be used as a reference.

Rules

1. To make this project more interesting, you are not allowed to use libFLAC, libFLAC++, ffmpeg, or any other library making it TOO easy to implement your convertor. Since your workload will be evaluated for grading.
2. The project files must be submitted before the deadline. Any submission after the deadline (even by 1 second) will result in a score of 0. The deadline is 23:59, on the Sunday of week 18.

Utilities

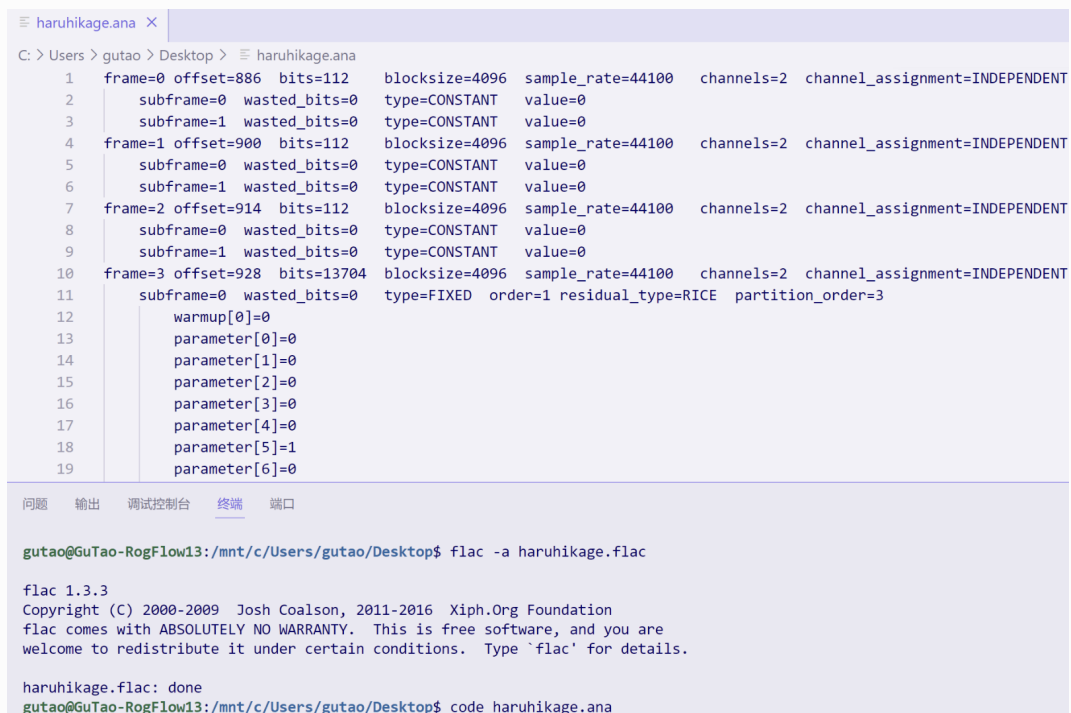
Heres some useful tools to help you understand the formats and file streams. Notice that you may **NOT** invoke them in your own source code.

- **FLAC**

The official command-line tools, which contains two sub programs. `flac` is an encoder as well as a decoder, while `metaflac` is a metadata editor. Your project may imitate the CLI of this official tool.

We highly recommend you to check the format of .flac files using `flac -a` rather than reading the file in hexadecimal when debugging.

```
flac input.wav      # encode WAVE into FLAC
flac -d input.flac  # decode FLAC into WAVE
flac -a input.flac  # analyze audio frames
```



The screenshot shows a code editor with a file named `haruhikage.ana` open. The file contains 19 lines of text representing FLAC metadata. The first 10 lines describe three frames of audio data with various parameters like `offset`, `bits`, `blocksize`, `sample_rate`, `channels`, and `channel_assignment`. Lines 11-19 describe the `warmup` and `parameter` sections. Below the code editor, a terminal window is open, showing the command `flac -a haruhikage.flac` being executed. The terminal output displays the version `flac 1.3.3`, copyright information, and a confirmation that the analysis is done.

```
haruhikage.ana
1  frame=0 offset=886 bits=112 blocksize=4096 sample_rate=44100 channels=2 channel_assignment=INDEPENDENT
2      subframe=0 wasted_bits=0 type=CONSTANT value=0
3      subframe=1 wasted_bits=0 type=CONSTANT value=0
4  frame=1 offset=900 bits=112 blocksize=4096 sample_rate=44100 channels=2 channel_assignment=INDEPENDENT
5      subframe=0 wasted_bits=0 type=CONSTANT value=0
6      subframe=1 wasted_bits=0 type=CONSTANT value=0
7  frame=2 offset=914 bits=112 blocksize=4096 sample_rate=44100 channels=2 channel_assignment=INDEPENDENT
8      subframe=0 wasted_bits=0 type=CONSTANT value=0
9      subframe=1 wasted_bits=0 type=CONSTANT value=0
10 frame=3 offset=928 bits=13704 blocksize=4096 sample_rate=44100 channels=2 channel_assignment=INDEPENDENT
11     subframe=0 wasted_bits=0 type=FIXED order=1 residual_type=RICE partition_order=3
12     warmup[0]=0
13     parameter[0]=0
14     parameter[1]=0
15     parameter[2]=0
16     parameter[3]=0
17     parameter[4]=0
18     parameter[5]=1
19     parameter[6]=0

gutao@GuTao-RogFlow13:/mnt/c/Users/gutao/Desktop$ flac -a haruhikage.flac

flac 1.3.3
Copyright (C) 2000-2009 Josh Coalson, 2011-2016 Xiph.Org Foundation
flac comes with ABSOLUTELY NO WARRANTY. This is free software, and you are
welcome to redistribute it under certain conditions. Type `flac' for details.

haruhikage.flac: done
gutao@GuTao-RogFlow13:/mnt/c/Users/gutao/Desktop$ code haruhikage.ana
```

- [Sound eXchange](#)

SoX is the Swiss Army Knife of sound processing utilities. It can convert audio files to other popular audio file types and also apply sound effects and filters during the conversion.

You may check if your convertor generates the same result as SoX does.

```
sox -i input.wav          # show WAVE header info
sox input.wav -t raw output.raw # decode WAVE into RAW
sox -t raw -c 2 -e signed-integer -b 16 -r 44100 name.raw name.wav
    # add headers for RAW and then encode them into WAVE
```

- [Hex Editor](#)

A custom editor extension for Visual Studio Code which provides a hex editor for viewing and manipulating files in their raw hexadecimal representation.

Reference

[Wikipedia: Waveform Audio File Format](#)

[Wikipedia: Free Lossless Audio Codec](#)

[FLAC Official Webpage](#)

[GitHub: xiph/flac](#)

[FLAC decoder testbench](#)