Este é o CS50

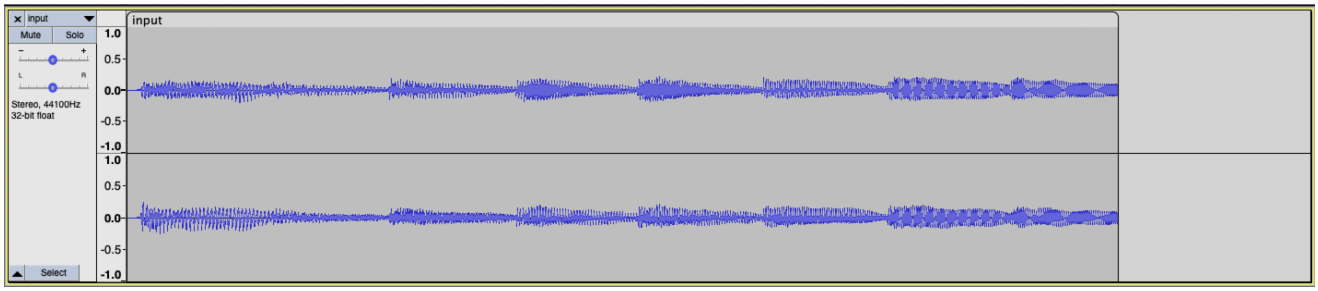
Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (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://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Volume



Problema a resolver

Os arquivos WAV (https://docs.fileformat.com/audio/wav/) são um formato comum para representar áudio. Eles armazenam áudio como uma sequência de "amostras": números que representam o valor de um sinal de áudio em um determinado instante. Os arquivos WAV começam com um "cabeçalho" de 44 bytes que contém informações sobre o próprio arquivo, incluindo o tamanho do arquivo, o número de amostras por segundo e o tamanho de cada amostra. Após o cabeçalho, o arquivo WAV contém uma sequência de amostras, cada uma um único inteiro de 2 bytes (16 bits) representando o sinal de áudio em um determinado instante.

Aumentar a escala de cada valor de amostra por um determinado fator tem o efeito de alterar o volume do áudio. Multiplicar cada valor de amostra por 2,0, por exemplo, terá o efeito de dobrar o volume do áudio original. Multiplicar cada amostra por 0,5, por sua vez, terá o efeito de reduzir o volume pela metade.

Em um arquivo chamado `audiofile` localizado `volume.c` em uma pasta chamada `audiofile` `volume`, escreva um programa para modificar o volume de um arquivo de áudio.

Demonstração

```
$ make volume
$ ./volume input.wav output.wav 2.0
$ ./volume input.wav output.wav 0.5
$ ./volume
Usage: ./volume input.wav output.wav factor
$ ./volume input.mp3 output.wav 2.0
Could not open file.
$
```

Recorded with **asciinema**

Código de Distribuição

Para este problema, você deverá estender a funcionalidade do código fornecido pela equipe do CS50.

▼ **Baixe o código de distribuição.**

Faça login em cs50.dev (https://cs50.dev/) , clique na janela do terminal e execute `cd` o comando. Você verá que o prompt do terminal será semelhante ao seguinte:

```
$
```

Em seguida, execute

```
wget https://cdn.cs50.net/2024/fall/psets/4/volume.zip
```

para baixar um arquivo ZIP chamado `volume.zip` para o seu espaço de código.

Em seguida, execute

```
unzip volume.zip
```

para criar uma pasta chamada `volume`. Você não precisa mais do arquivo ZIP, então pode executar

```
rm volume.zip
```

e responda com “y” seguido de Enter quando solicitado para remover o arquivo ZIP que você baixou.

Agora digite

```
cd volume
```

Em seguida, pressione Enter para entrar (ou seja, abrir) esse diretório. Seu prompt agora deve ser semelhante ao abaixo.

```
volume/ $
```

Se tudo correr bem, você deverá executar

```
ls
```

e veja um arquivo chamado `volume.c`. A execução `code volume.c` deve abrir o arquivo onde você digitará o código para este conjunto de problemas. Caso contrário, refaça seus passos e veja se consegue determinar onde errou!

Detalhes da implementação

Complete a implementação de `volume.c`, de forma que ela altere o volume de um arquivo de som por um fator determinado.

- O programa deve aceitar três argumentos de linha de comando. O primeiro é `<nome_do_arquivo_de_áudio>` `input`, que representa o nome do arquivo de áudio original. O segundo é `<nome_do_novo_arquivo_de_áudio>` `output`, que representa o nome do novo arquivo de áudio a ser gerado. O terceiro é `<valor_do_volume>` `factor`, que é o valor pelo qual o volume do arquivo de áudio original deve ser aumentado.
  - Por exemplo, se `factor` for `2.0`, então seu programa deve dobrar o volume do arquivo de áudio em `input` e salvar o arquivo de áudio recém-gerado em `output`.
- Seu programa deve primeiro ler o cabeçalho do arquivo de entrada e escrever o cabeçalho no arquivo de saída.
- Seu programa deve então ler o restante dos dados do arquivo WAV, uma amostra de 16 bits (2 bytes) por vez. Seu programa deve multiplicar cada amostra pelo valor de 16 bits `factor` e escrever a nova amostra no arquivo de saída.
  - Você pode assumir que o arquivo WAV usará valores de 16 bits com sinal como amostras. Na prática, os arquivos WAV podem ter números variáveis de bits por amostra, mas assumiremos amostras de 16 bits para este problema.
- Seu programa, caso utilize `malloc`, não deve apresentar vazamento de memória.

Dicas

Clique nos botões abaixo para ler algumas dicas!

▼ **Entenda o código em `volume.c`**

Observe primeiro que `volume.c` já está configurado para receber três argumentos de linha de comando, `input`, `output`, e `factor`.

- `main` recebe um `int`, `argc`, e um array de `char *`s (strings!), `argv`.
- Se `argc` o número de argumentos na linha de comando, incluindo o próprio programa, for diferente de 4, o programa exibirá seu uso correto e será encerrado com o código de status 1.

```
int main(int argc, char *argv[])
{
    // Check command-line arguments
    if (argc != 4)
    {
        printf("Usage: ./volume input.wav output.wav factor\n");
        return 1;
    }

    // ...
}
```

Em seguida, `volume.c` utiliza-se o comando `fopen` (https://manual.cs50.io/3/fopen)para abrir os dois arquivos fornecidos como argumentos da linha de comando.

- É uma boa prática verificar se o resultado da chamada `fopen` é `NULL`. Se for, o arquivo não foi encontrado ou não pôde ser aberto.

```
// Open files and determine scaling factor
FILE *input = fopen(argv[1], "r");
if (input == NULL)
{
    printf("Could not open file.\n");
    return 1;
}

FILE *output = fopen(argv[2], "w");
if (output == NULL)
{
    printf("Could not open file.\n");
    return 1;
}
```

Posteriormente, esses arquivos são fechados com `fclose`. Sempre que você chamar `fopen`, você deverá chamar em seguida `fclose`!

```
// Close files
fclose(input);
fclose(output);
```

Antes de fechar os arquivos, porém, observe que temos algumas tarefas pendentes.

```
// TODO: Copy header from input file to output file

// TODO: Read samples from input file and write updated data to output file
```

É bem provável que você precise saber o fator pelo qual dimensionar o volume, por isso o programa `volume.c` já converte o terceiro argumento da linha de comando em um valor `float` para você!

```
float factor = atof(argv[3]);
```

▼ **Copiar o cabeçalho WAV do arquivo de entrada para o arquivo de saída.**

Sua primeira tarefa é copiar o cabeçalho do arquivo WAV `input` e gravá-lo em `output`. Primeiro, porém, você precisará aprender sobre alguns tipos de dados especiais.

Até agora, vimos vários tipos diferentes em C, incluindo `int` `int`, `bool` `int`, `char` `int`, `double` `int`, `float` `int` e ` int` `long`. No entanto, dentro de um arquivo de cabeçalho chamado `std ::string` `stdint.h`, encontramos as declarações de vários *outros* tipos que nos permitem definir com muita precisão o tamanho (em bits) e o sinal (com ou sem sinal) de um inteiro. Dois tipos em particular serão úteis para nós ao trabalhar com arquivos WAV:

- `uint8_t` `8` é um tipo que armazena um inteiro sem sinal (ou seja, não negativo) de 8 bits (daí o ! `uint`). Podemos tratar cada byte do cabeçalho de um arquivo WAV como um `uint8_t` valor.
- `int16_t` é um tipo que armazena um inteiro de 16 bits com sinal (ou seja, positivo ou negativo). Podemos tratar cada amostra de áudio em um arquivo WAV como um `int16_t` valor.

Provavelmente você vai querer criar uma matriz de bytes para armazenar os dados do cabeçalho do arquivo WAV que serão lidos do arquivo de entrada. Usando o `uint8_t` tipo para representar um byte, você pode criar uma matriz de `n` bytes para o seu cabeçalho com a seguinte sintaxe:

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

substituindo `n` pelo número de bytes. Você pode então usar `header` como argumento para `fread` (https://manual.cs50.io/3/fread)ler `fwrite` (https://manual.cs50.io/3/fwrite)ou escrever no cabeçalho.

Lembre-se de que o cabeçalho de um arquivo WAV tem sempre exatamente 44 bytes de comprimento. Observe que `volume.c` já existe uma variável definida para você, chamada

`HEADER_SIZE`, igual ao número de bytes no cabeçalho.

A seguir, uma dica bastante importante, mas aqui está como você pode realizar esta tarefa!

```
// Copy header from input file to output file
uint8_t header[HEADER_SIZE];
fread(header, HEADER_SIZE, 1, input);
fwrite(header, HEADER_SIZE, 1, output);
```

▼ **Escrever dados atualizados no arquivo de saída**

Sua próxima tarefa é ler amostras de `input`, atualizar essas amostras e gravar as amostras atualizadas em `output`. Ao ler arquivos, é comum criar um "buffer" para armazenar dados temporariamente. Nele, você pode modificar os dados e, quando estiverem prontos, gravar os dados do buffer em um novo arquivo.

Lembre-se de que podemos usar o `int16_t` tipo para representar uma amostra de um arquivo WAV. Para armazenar uma amostra de áudio, você pode criar uma variável de buffer com uma sintaxe como esta:

```
// Create a buffer for a single sample
int16_t buffer;
```

Com um buffer para amostras configurado, agora você pode ler dados para ele, uma amostra por vez. Experimente usar ` std::string` `fread` para essa tarefa! Você pode usar `std:: `&buffer` string`, o endereço de `std::string` `buffer`, como argumento para `fread` ` std::string` ou `std::vector` `fwrite` para ler ou escrever no buffer. (Lembre-se de que o `&` operador `&` é usado para obter o endereço da variável.)

```
// Create a buffer for a single sample
int16_t buffer;

// Read single sample into buffer
fread(&buffer, sizeof(int16_t), 1, input);
```

Agora, para aumentar (ou diminuir) o volume de uma amostra, basta multiplicá-lo por algum fator.

```
// Create a buffer for a single sample
int16_t buffer;

// Read single sample into buffer
fread(&buffer, sizeof(int16_t), 1, input);

// Update volume of sample
buffer *= factor;
```

E, finalmente, você pode escrever esse exemplo atualizado em `output`:

```
// Create a buffer for a single sample
int16_t buffer;

// Read single sample from input into buffer
fread(&buffer, sizeof(int16_t), 1, input);

// Update volume of sample
buffer *= factor;

// Write updated sample to new file
fwrite(&buffer, sizeof(int16_t), 1, output);
```

Há apenas um problema: você precisará *continuar* lendo uma amostra para o seu buffer, atualizando seu volume e gravando a amostra atualizada no arquivo de saída enquanto ainda houver amostras para ler.

- Felizmente, conforme a documentação, o `fread` comando retornará o número de itens de dados lidos com sucesso. Isso pode ser útil para verificar quando você chegou ao final do arquivo!
- Lembre-se de que não há motivo para você não poder fazer uma chamada `fread` dentro da `while` condicional de um loop. Você poderia, por exemplo, fazer uma chamada `fread` como a seguinte:

  ```
  while (fread(...))
  {

  }
  ```

É uma dica bastante sutil, mas veja abaixo uma maneira eficiente de resolver esse problema:

```
// Create a buffer for a single sample
int16_t buffer;

// Read single sample from input into buffer while there are samples left to read
while (fread(&buffer, sizeof(int16_t), 1, input) != 0)
{
    // Update volume of sample
    buffer *= factor;

    // Write updated sample to new file
    fwrite(&buffer, sizeof(int16_t), 1, output);
}
```

Como a versão de C que você está usando trata valores diferentes de zero como `NULL` `true` e valores zero como `NULL` `false`, você pode simplificar a sintaxe acima para o seguinte:

```
// Create a buffer for a single sample
int16_t buffer;

// Read single sample from input into buffer while there are samples left to read
while (fread(&buffer, sizeof(int16_t), 1, input))
{
```

```
    // Update volume of sample
    buffer *= factor;

    // Write updated sample to new file
    fwrite(&buffer, sizeof(int16_t), 1, output);
}
```

Passo a passo

▶ **Não sabe como resolver?**

Como testar

Seu programa deve se comportar conforme os exemplos abaixo.

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

Ao ouvir o arquivo `output.wav` (por exemplo, clicando com a tecla Control pressionada `output.wav` no arquivo no explorador de arquivos, escolhendo "**Download**" e, em seguida, abrindo o arquivo em um reprodutor de áudio no seu computador), o volume deve ser duas vezes maior que o original `input.wav`!

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

Ao ouvir `output.wav`, o volume deve ser metade do volume de `input.wav`!

Correção

```
check50 cs50/problems/2025/x/volume
```

Estilo

```
style50 volume.c
```

Como enviar

```
submit50 cs50/problems/2025/x/volume
```