

# 视音频数据处理入门：PCM音频采样数据处理

2016年01月29日 23:32:13 阅读数：71906

视音频数据处理入门系列文章：

[视音频数据处理入门：RGB、YUV像素数据处理](#)

[视音频数据处理入门：PCM音频采样数据处理](#)

[视音频数据处理入门：H.264视频码流解析](#)

[视音频数据处理入门：AAC音频码流解析](#)

[视音频数据处理入门：FLV封装格式解析](#)

[视音频数据处理入门：UDP-RTP协议解析](#)

上一篇文章记录了RGB/YUV视频像素数据的处理方法，本文继续上一篇文章的内容，记录PCM音频采样数据的处理方法。音频采样数据在视频播放器的解码流程中的位置如下图所示。



本文分别介绍如下几个PCM音频采样数据处理函数：

分离PCM16LE双声道音频采样数据的左声道和右声道

将PCM16LE双声道音频采样数据中左声道的音量降一半

将PCM16LE双声道音频采样数据的声音速度提高一倍

将PCM16LE双声道音频采样数据转换为PCM8音频采样数据

从PCM16LE单声道音频采样数据中截取一部分数据

将PCM16LE双声道音频采样数据转换为WAVE格式音频数据

注：PCM音频数据可以使用音频编辑软件导入查看。例如收费的专业音频编辑软件 [Adobe Audition](#)，或者免费开源的音频编辑软件 [Audacity](#)。

## 函数列表

### (1) 分离PCM16LE双声道音频采样数据的左声道和右声道

本程序中的函数可以将PCM16LE双声道数据中左声道和右声道的数据分离成两个文件。函数的代码如下所示。

```
[cpp]
1.  /**
2.   * Split Left and Right channel of 16LE PCM file.
3.   * @param url Location of PCM file.
4.   *
5.   */
6.  int simplest_pcm16le_split(char *url){
7.      FILE *fp=fopen(url,"rb+");
8.      FILE *fp1=fopen("output_l.pcm","wb+");
9.      FILE *fp2=fopen("output_r.pcm","wb+");
10.
11.      unsigned char *sample=(unsigned char *)malloc(4);
12.
13.      while(!feof(fp)){
14.          fread(sample,1,4,fp);
15.          //L
16.          fwrite(sample,1,2,fp1);
17.          //R
18.          fwrite(sample+2,1,2,fp2);
19.      }
20.
21.      free(sample);
22.      fclose(fp);
23.      fclose(fp1);
24.      fclose(fp2);
25.      return 0;
26.  }
```

调用上面函数的方法如下所示。

```
[cpp]
1.  simplest_pcm16le_split("NocturneNo2inEflat_44.1k_s16le.pcm");
```

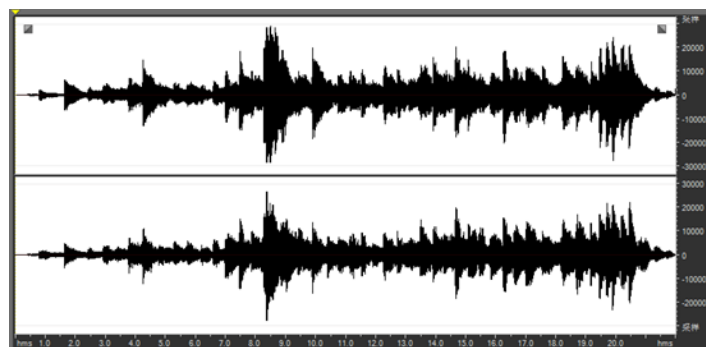
从代码可以看出，PCM16LE双声道数据中左声道和右声道的采样值是间隔存储的。每个采样值占用2Byte空间。代码运行后，会把NocturneNo2inEflat\_44.1k\_s16le.pcm的PCM16LE格式的数据分离为两个单声道数据：

output\_l.pcm：左声道数据。

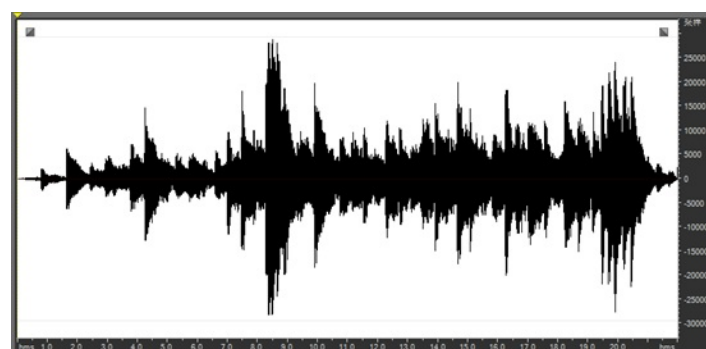
output\_r.pcm：右声道数据。

注：本文中声音样值的采样频率一律是44100Hz，采样格式一律为16LE。“16”代表采样位数是16bit。由于1Byte=8bit，所以一个声道的一个采样值占用2Byte。“LE”代表Little Endian，代表2 Byte采样值的存储方式为高位存在高地址中。

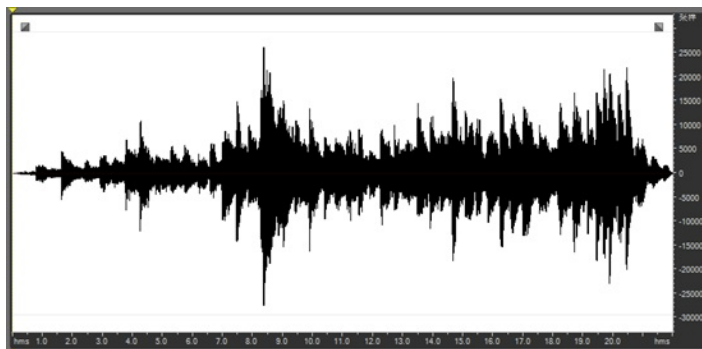
下图为输入的双声道PCM数据的波形图。上面的波形图是左声道的图形，下面的波形图是右声道的波形。图中的横坐标是时间，总长度为22秒；纵坐标是取样值，取值范围从-32768到32767。



下图为分离后左声道数据output\_l.pcm的音频波形图。



下图为分离后右声道数据output\_r.pcm的音频波形图。



## (2) 将PCM16LE双声道音频采样数据中左声道的音量降一半

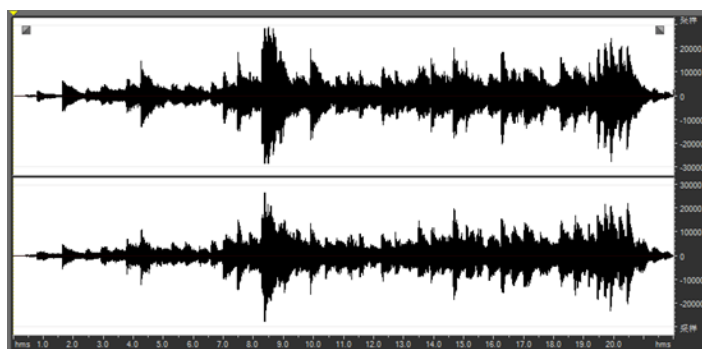
本程序中的函数可以将PCM16LE双声道数据中左声道的音量降低一半。函数的代码如下所示。

```
[cpp]
1.  /**
2.   * Halve volume of Left channel of 16LE PCM file
3.   * @param url Location of PCM file.
4.   */
5.  int simplest_pcm16le_halfvolumeleft(char *url){
6.      FILE *fp=fopen(url,"rb+");
7.      FILE *fp1=fopen("output_halfleft.pcm","wb+");
8.
9.      int cnt=0;
10.
11.      unsigned char *sample=(unsigned char *)malloc(4);
12.
13.      while(!feof(fp)){
14.          short *samplenum=NULL;
15.          fread(sample,1,4,fp);
16.
17.          samplenum=(short *)sample;
18.          *samplenum=*samplenum/2;
19.          //L
20.          fwrite(sample,1,2,fp1);
21.          //R
22.          fwrite(sample+2,1,2,fp1);
23.
24.          cnt++;
25.      }
26.      printf("Sample Cnt:%d\n",cnt);
27.
28.      free(sample);
29.      fclose(fp);
30.      fclose(fp1);
31.      return 0;
32.  }
```

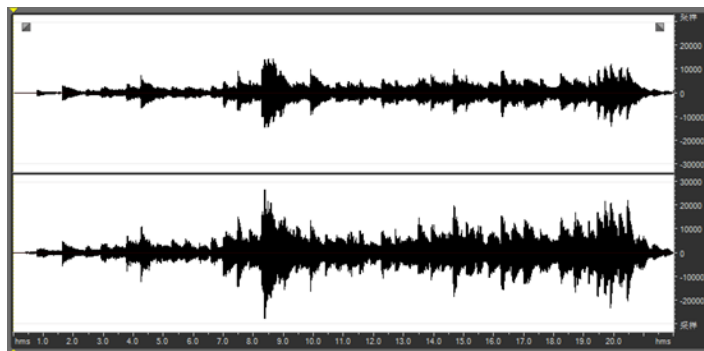
调用上面函数的方法如下所示。

```
[cpp]
1.  simplest_pcm16le_halfvolumeleft("NocturneNo2inEflat_44.1k_s16le.pcm");
```

从源代码可以看出,本程序在读出左声道的2 Byte的取样值之后,将其当成了C语言中的一个short类型的变量。将该数值除以2之后写回到了PCM文件中。下图为输入PCM双声道音频采样数据的波形图。



下图为输出的左声道经过处理后的波形图。可以看出左声道的波形幅度降低了一半。



### (3) 将PCM16LE双声道音频采样数据的声音速度提高一倍

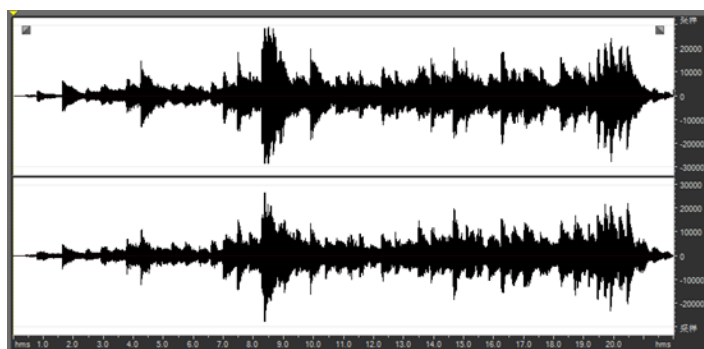
本程序中的函数可以通过抽象的方式将PCM16LE双声道数据的速度提高一倍。函数的代码如下所示。

```
[cpp]
1.  /**
2.   * Re-sample to double the speed of 16LE PCM file
3.   * @param url Location of PCM file.
4.   */
5.  int simplest_pcm16le_doublespeed(char *url){
6.      FILE *fp=fopen(url,"rb+");
7.      FILE *fp1=fopen("output_doublespeed.pcm","wb+");
8.
9.      int cnt=0;
10.
11.      unsigned char *sample=(unsigned char *)malloc(4);
12.
13.      while(!feof(fp)){
14.
15.          fread(sample,1,4,fp);
16.
17.          if(cnt%2!=0){
18.              //L
19.              fwrite(sample,1,2,fp1);
20.              //R
21.              fwrite(sample+2,1,2,fp1);
22.          }
23.          cnt++;
24.      }
25.      printf("Sample Cnt:%d\n",cnt);
26.
27.      free(sample);
28.      fclose(fp);
29.      fclose(fp1);
30.      return 0;
31.  }
```

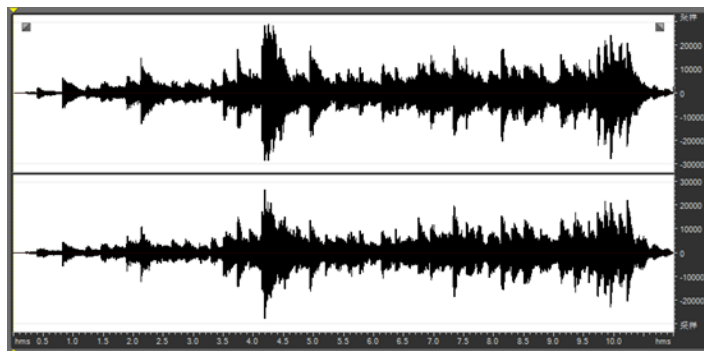
调用上面函数的方法如下所示。

```
[cpp]
1.  simplest_pcm16le_doublespeed("NocturneNo2inEflat_44.1k_s16le.pcm");
```

从源代码可以看出，本程序只采样了每个声道奇数点的样值。处理完成后，原本22秒左右的音频变成了11秒左右。音频的播放速度提高了2倍，音频的音调也变高了很多。下图为输入PCM双声道音频采样数据的波形图。



下图为输出的PCM双声道音频采样数据的波形图。通过时间轴可以看出音频变短了很多。



#### (4) 将PCM16LE双声道音频采样数据转换为PCM8音频采样数据

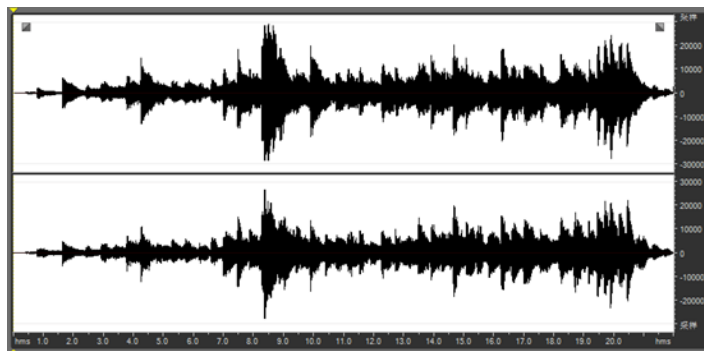
本程序中的函数可以通过计算的方式将PCM16LE双声道数据16bit的采样位数转换为8bit。函数的代码如下所示。

```
[cpp]
1.  /**
2.   * Convert PCM-16 data to PCM-8 data.
3.   * @param url Location of PCM file.
4.   */
5.  int simplest_pcm16le_to_pcm8(char *url){
6.      FILE *fp=fopen(url,"rb+");
7.      FILE *fp1=fopen("output_8.pcm","wb+");
8.
9.      int cnt=0;
10.
11.      unsigned char *sample=(unsigned char *)malloc(4);
12.
13.      while(!feof(fp)){
14.
15.          short *samplenum16=NULL;
16.          char samplenum8=0;
17.          unsigned char samplenum8_u=0;
18.          fread(sample,1,4,fp);
19.          //(-32768-32767)
20.          samplenum16=(short *)sample;
21.          samplenum8=(*samplenum16)>>8;
22.          //(0-255)
23.          samplenum8_u=samplenum8+128;
24.          //L
25.          fwrite(&samplenum8_u,1,1,fp1);
26.
27.          samplenum16=(short *) (sample+2);
28.          samplenum8=(*samplenum16)>>8;
29.          samplenum8_u=samplenum8+128;
30.          //R
31.          fwrite(&samplenum8_u,1,1,fp1);
32.          cnt++;
33.      }
34.      printf("Sample Cnt:%d\n",cnt);
35.
36.      free(sample);
37.      fclose(fp);
38.      fclose(fp1);
39.      return 0;
40.  }
```

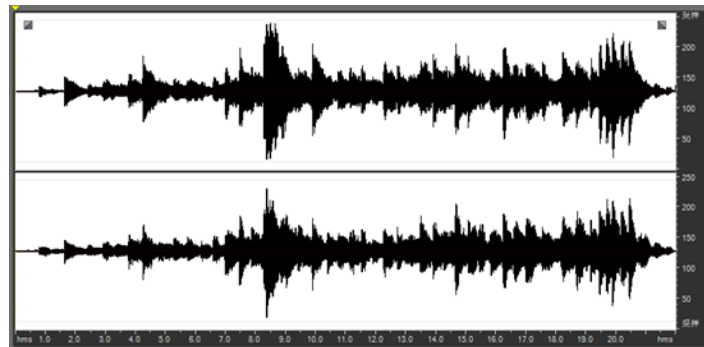
调用上面函数的方法如下所示。

```
[cpp]
1.  simplest_pcm16le_to_pcm8("NocturneNo2inEflat_44.1k_s16le.pcm");
```

PCM16LE格式的采样数据的取值范围是-32768到32767，而PCM8格式的采样数据的取值范围是0到255。所以PCM16LE转换到PCM8需要经过两个步骤：第一步是将-32768到32767的16bit有符号数值转换为-128到127的8bit有符号数值，第二步是将-128到127的8bit有符号数值转换为0到255的8bit无符号数值。在本程序中，16bit采样数据是通过short类型变量存储的，而8bit采样数据是通过unsigned char类型存储的。下图为输入的16bit的PCM双声道音频采样数据的波形图。



下图为输出的8bit的PCM双声道音频采样数据的波形图。注意观察图中纵坐标的取值范围已经变为0至255。如果发现8bit PCM的音质明显不如16 bit PCM的音质。



## (5) 将从PCM16LE单声道音频采样数据中截取一部分数据

本程序中的函数可以从PCM16LE单声道数据中截取一段数据，并输出截取数据的样值。函数的代码如下所示。

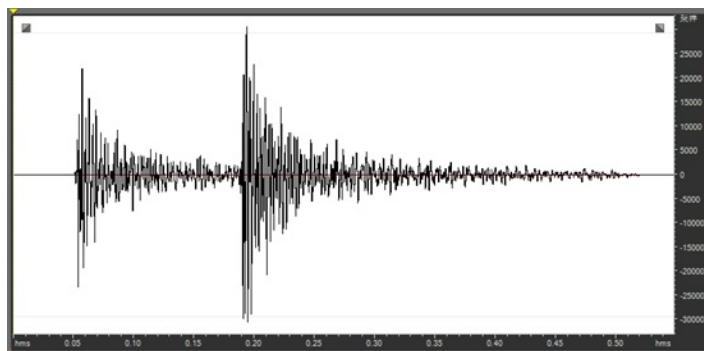
```
[cpp]
1.  /**
2.   * Cut a 16LE PCM single channel file.
3.   * @param url      Location of PCM file.
4.   * @param start_num start point
5.   * @param dur_num  how much point to cut
6.   */
7.  int simplest_pcm16le_cut_singlechannel(char *url,int start_num,int dur_num){
8.      FILE *fp=fopen(url,"rb+");
9.      FILE *fp1=fopen("output_cut.pcm","wb+");
10.     FILE *fp_stat=fopen("output_cut.txt","wb+");
11.
12.     unsigned char *sample=(unsigned char *)malloc(2);
13.
14.     int cnt=0;
15.     while(!feof(fp)){
16.         fread(sample,1,2,fp);
17.         if(cnt>start_num&&cnt<=(start_num+dur_num)){
18.             fwrite(sample,1,2,fp1);
19.
20.             short samplenum=sample[1];
21.             samplenum=samplenum*256;
22.             samplenum=samplenum+sample[0];
23.
24.             fprintf(fp_stat,"%6d",samplenum);
25.             if(cnt%10==0)
26.                 fprintf(fp_stat,"\n",samplenum);
27.         }
28.         cnt++;
29.     }
30.
31.     free(sample);
32.     fclose(fp);
33.     fclose(fp1);
34.     fclose(fp_stat);
35.     return 0;
36. }
```

调用上面函数的方法如下所示。

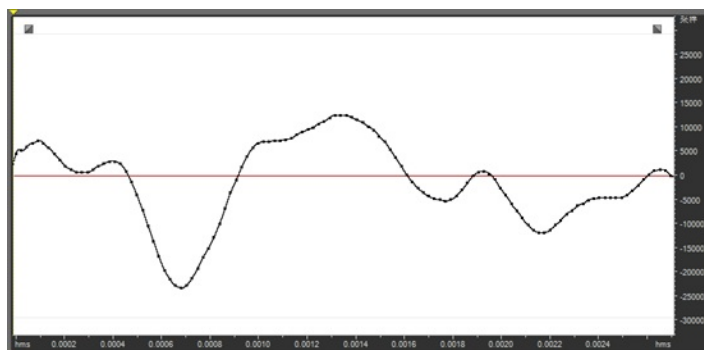
```
[cpp]
1.  simplest_pcm16le_cut_singlechannel("drum.pcm",2360,120);
```

本程序可以从PCM数据中选取一段采样值保存下来，并且输出这些采样值的数值。上述代码运行后，会把单声道PCM16LE格式的“drum.pcm”中从

2360点开始的120点的数据保存成output\_cut.pcm文件。下图为“drum.pcm”的波形图，该音频采样频率为44100KHz，长度为0.5秒，一共包含约22050个采样点。



下图为截取出来的output\_cut.pcm文件中的数据。



下面列出了上述数据的采样值。

```
[plain]
1. 4460, 5192, 5956, 6680, 7199, 6706, 5727, 4481, 3261, 1993,
2. 1264, 747, 767, 752, 1248, 1975, 2473, 2955, 2952, 2447,
3. 974, -1267, -4000, -6965, -10210, -13414, -16639, -19363, -21329, -22541,
4. 23028, -22545, -21055, -19067, -16829, -14859, -12596, -9900, -6684, -3475,
5. -983, 1733, 3978, 5734, 6720, 6978, 6993, 7223, 7225, 7440,
6. 7688, 8431, 8944, 9468, 9947, 10688, 11194, 11946, 12449, 12446,
7. 12456, 11974, 11454, 10952, 10167, 9425, 8153, 6941, 5436, 3716,
8. 1952, 236, -1254, -2463, -3493, -4223, -4695, -4927, -5190, -4941,
9. -4188, -2956, -1490, -40, 705, 932, 446, -776, -2512, -3994,
10. -5723, -7201, -8687, -10157, -11134, -11661, -11642, -11168, -10155, -9142,
11. -7888, -7146, -6186, -5694, -4971, -4715, -4498, -4471, -4468, -4452,
12. -4452, -3940, -2980, -1984, -752, 257, 1021, 1264, 1032, 31,
```

## (6) 将PCM16LE双声道音频采样数据转换为WAVE格式音频数据

WAVE格式音频（扩展名为“.wav”）是Windows系统中最常见的一种音频。该格式的实质就是在PCM文件的前面加了一个文件头。本程序的函数就可以通过在PCM文件前面加一个WAVE文件头从而封装为WAVE格式音频。函数的代码如下所示。

```
[cpp]
1. /**
2.  * Convert PCM16LE raw data to WAVE format
3.  * @param pcm_path Input PCM file.
4.  * @param channels Channel number of PCM file.
5.  * @param sample_rate Sample rate of PCM file.
6.  * @param wave_path Output WAVE file.
7.  */
8. int simplest_pcm16le_to_wave(const char *pcm_path, int channels, int sample_rate, const char *wave_path)
9. {
10.
11.     typedef struct WAVE_HEADER{
12.         char fccID[4];
13.         unsigned long dwSize;
14.         char fccType[4];
15.     }WAVE_HEADER;
16.
17.     typedef struct WAVE_FMT{
18.         char fccID[4];
19.         unsigned long dwSize;
20.         unsigned short wFormatTag;
21.         unsigned short wChannels;
22.         unsigned long dwSamplesPerSec;
23.         unsigned long dwAvgBytesPerSec;
24.         unsigned short wBlockAlign;
25.         unsigned short uiBitsPerSample;
26.     }WAVE_FMT;
```

```

27.
28.     typedef struct WAVE_DATA{
29.         char        fccID[4];
30.         unsigned long dwSize;
31.     }WAVE_DATA;
32.
33.
34.     if(channels==0||sample_rate==0){
35.         channels = 2;
36.         sample_rate = 44100;
37.     }
38.     int bits = 16;
39.
40.     WAVE_HEADER  pcmHEADER;
41.     WAVE_FMT      pcmFMT;
42.     WAVE_DATA     pcmDATA;
43.
44.     unsigned short m_pcmData;
45.     FILE *fp,*fpout;
46.
47.     fp=fopen(pcmopath, "rb");
48.     if(fp == NULL) {
49.         printf("open pcm file error\n");
50.         return -1;
51.     }
52.     fpout=fopen(wavepath, "wb+");
53.     if(fpout == NULL) {
54.         printf("create wav file error\n");
55.         return -1;
56.     }
57.     //WAVE_HEADER
58.     memcpy(pcmHEADER.fccID,"RIFF",strlen("RIFF"));
59.     memcpy(pcmHEADER.fccType,"WAVE",strlen("WAVE"));
60.     fseek(fpout,sizeof(WAVE_HEADER),1);
61.     //WAVE_FMT
62.     pcmFMT.dwSamplesPerSec=sample_rate;
63.     pcmFMT.dwAvgBytesPerSec=pcmFMT.dwSamplesPerSec*sizeof(m_pcmData);
64.     pcmFMT.wBitsPerSample=bits;
65.     memcpy(pcmFMT.fccID,"fmt ",strlen("fmt "));
66.     pcmFMT.dwSize=16;
67.     pcmFMT.wBlockAlign=2;
68.     pcmFMT.wChannels=channels;
69.     pcmFMT.wFormatTag=1;
70.
71.     fwrite(&pcmFMT,sizeof(WAVE_FMT),1,fpout);
72.
73.     //WAVE_DATA;
74.     memcpy(pcmDATA.fccID,"data",strlen("data"));
75.     pcmDATA.dwSize=0;
76.     fseek(fpout,sizeof(WAVE_DATA),SEEK_CUR);
77.
78.     fread(&m_pcmData,sizeof(unsigned short),1,fp);
79.     while(!feof(fp)){
80.         pcmDATA.dwSize+=2;
81.         fwrite(&m_pcmData,sizeof(unsigned short),1,fpout);
82.         fread(&m_pcmData,sizeof(unsigned short),1,fp);
83.     }
84.
85.     pcmHEADER.dwSize=44+pcmDATA.dwSize;
86.
87.     rewind(fpout);
88.     fwrite(&pcmHEADER,sizeof(WAVE_HEADER),1,fpout);
89.     fseek(fpout,sizeof(WAVE_FMT),SEEK_CUR);
90.     fwrite(&pcmDATA,sizeof(WAVE_DATA),1,fpout);
91.
92.     fclose(fp);
93.     fclose(fpout);
94.
95.     return 0;
96. }

```

调用上面函数的方法如下所示。

```

1.  simplest_pcm16le_to_wave("NocturneNo2inEflat_44.1k_s16le.pcm",2,44100,"output_nocturne.wav");



```

WAVE文件是一种RIFF格式的文件。其基本块名称是“WAVE”，其中包含了两个子块“fmt”和“data”。从编程的角度简单说来说就是由WAVE\_HEADER、WAVE\_FMT、WAVE\_DATA、采样数据共4部分组成。它的结构如下所示。

WAVE_HEADER
WAVE_FMT
WAVE_DATA
PCM数据



其中前3部分的结构如下所示。在写入WAVE文件头的时候给其中的每个字段赋上合适的值就可以了。但是有一点需要注意：WAVE\_HEADER和WAVE\_DATA中包含了一个文件长度信息的dwSize字段，该字段的值必须在写完音频采样数据之后才能获得。因此这两个结构体最后才写入WAVE文件中。

```
[cpp]    
1.  typedef struct WAVE_HEADER{  
2.      char fccID[4];  
3.      unsigned long dwSize;  
4.      char fccType[4];  
5.  }WAVE_HEADER;  
6.  
7.  typedef struct WAVE_FMT{  
8.      char fccID[4];  
9.      unsigned long dwSize;  
10.     unsigned short wFormatTag;  
11.     unsigned short wChannels;  
12.     unsigned long dwSamplesPerSec;  
13.     unsigned long dwAvgBytesPerSec;  
14.     unsigned short wBlockAlign;  
15.     unsigned short uiBitsPerSample;  
16.  }WAVE_FMT;  
17.  
18.  typedef struct WAVE_DATA{  
19.      char fccID[4];  
20.      unsigned long dwSize;  
21.  }WAVE_DATA;
```

本程序的函数执行完成后，就可将NocturneNo2inEflat\_44.1k\_s16le.pcm文件封装成output\_nocturne.wav文件。

## 下载

Simplest mediadata test

### 项目主页

SourceForge：<https://sourceforge.net/projects/simplest-mediadata-test/>

Github：[https://github.com/leixiaohua1020/simplest\\_mediadata\\_test](https://github.com/leixiaohua1020/simplest_mediadata_test)

开源中国：[http://git.oschina.net/leixiaohua1020/simplest\\_mediadata\\_test](http://git.oschina.net/leixiaohua1020/simplest_mediadata_test)

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/9422409>

本项目包含如下几种视音频数据解析示例：

- (1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
- (2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
- (3)H.264码流分析程序。可以分离并解析NALU。
- (4)AAC码流分析程序。可以分离并解析ADTS帧。
- (5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
- (6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。

**雷霄骅 (Lei Xiaohua)**

**leixiaohua1020@126.com**

**<http://blog.csdn.net/leixiaohua1020>**

文章标签：[PCM](#) [音频](#) [声道](#) [WAV](#)

个人分类：[我的开源项目](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com